help_ Subroutine

A replacement for the help subsystem.

Author: Gary Dixon
Date: February 13, 2021

Abstract

This MTB discusses reasons for replacing the Multics help subsystem (help_ and help_rql_ subroutines). It then proposes a new implementation for the existing help_ interface that uses the proposed info_seg_ routine to parse each info segment; and uses the Subsystem Utilities (ssu_) to coordinate the display of parsed info segments, and to invoke routines that implement "More help?" responses from the user.

Table 1: Revision History

Date	Revision	Author	Comment
2021-02-13	0.1	Gary Dixon	Initial draft.
2021-02-13	1.0	Gary Dixon	Response to comments from preliminary review by Eric Swenson.

Table of Contents

Abstract 1

List of Figures 4

List of Tables 4

Introduction 5

help_ & help_rql_: Issues with Current Code 5

Goals for help & help_ 6

Strategies for help_ 7

help Command Changes 9

help_ Subroutine Changes 11

help_: Use of ssu_ 11

help_$init Entry Point Changes 11

help_info (hi): A New Structure 13

help_info - Delayed-Allocation Storage Mechanism in the Temporary Segment 13

help_info - Operating Environment 17

help_info - Data for Found Info Blocks 18

help_info_ Data for Displaying Info Blocks 18

help_$help_ Entry Point Changes: progress = 1, 2 22

help_$help_ Entry Point Changes: progress = 3 22

help_$help_ Entry Point Changes: progress = 4 23

help_$help_ get_info_seg_list Procedure 23

help_$help_ Entry Point Changes: progress = 5 25

help_$help_ Replacement for help_rql_ 25

help_$help_ display_block Procedure 27

help_$term Entry Point Changes 30

help_ Shared Procedures: error, ioa, newline, response_error 31

ssu_ Listener 32

help_listen_util_ Subroutine 34

help Use-Cases 34

help_ State Information 36

help_listen_util_$display_prompt 38

help_listen_util_$set_iPgh_range 40

help_listen_util_$print_iPgh_range 41

help_responses_ Subroutine *42*
Response: all_paragraphs, all, all_entry_points, aep 45
Response: brief, bf 45
Response: control_arg, ca 46
Response: entry_point, entrypoint, ep 47
Response: every_paragraph_visible, every_pgh, every 47
Response: find_ep, find_info, find 48
Response: header, he 49
Response: info 50
Response: list_entry_points, list_ep, lep 51
help_responses_$lep_setup 51
Response: next, no, n 51
Response: rest, r 52
Response: section, scn 53
Response: search, srh 54
Response: skip, s 55
Response: titles, title 56
Response: top, t 57
Response: unseen_paragraphs_only, unseen, uns 58
Response: yes, y 58
help_requests_ Subroutine *59*
Response: quit, q 59
Response: . (self-identify) 59
Response: ? (summarize_responses) 59
Response: help, h 60
help_requests_$unknown_response 61
help_util_ Subroutine *63*
info_seg_ Subroutine Changes *65*
info_seg_$init_for_help_ 65
info_seg_$examine_iFile 66
info_seg_$reinitialize 67

info_seg_parse_ Subroutine Change ... ***68***

info_seg_parse_$append_STRING_to_block ... **68**

info_seg_parse_$replace_STRING_in_section ... **68**

Test Strategies ... ***69***

Command Interface Testing ... **69**

Subsystem Interface Testing ... **69**

help_ Subroutine Testing ... **70**

help_ Response Testing ... **71**

Installation Plan ... ***72***

MTBs ... **72**

MCRs ... **72**

Bugs (Multics Tickets) Not Resolved by these MTBs/MCRs ... **72**

Appendix A: Glossary ... ***73***

List of Figures

Figure 1: Temp Segment after calling help_$init ... 15
Figure 2: Temp Segment after help_$help_ progress = 3 ... 15
Figure 3: Temp Segment before help_$help_ calls display_block ... 16
Figure 4: Temp Segment when running display_block ... 16

List of Tables

Table 1: Revision History ... 1

Introduction

"MTB-1004: verify_info Command & info_seg_" proposes several enhancements to the format of info segments in order to properly verify their contents. From a help_ perspective, this proposal adds support for two new info block divider tokens.

- A "`:[Info]:`" token: designates an info block whose divider names should not appear as external names on the info segment file.
- A "`:hcom:`" token: to simplify addition of a history comment block at the end of an info segment.

The info_seg_ routines described in MTB-1004 already handle these enhancements. But the current help_ subsystem uses its own code to parse each info segment as it is displayed.

Support for these enhancements in help_ requires either:

A. a parallel change in the parsing code of help_ and help_rql_ code to support the new tokens; or
B. converting the existing help_ subsystem to use the new info_seg_ subroutine to parse an info segment into its components (a file containing blocks; each block beginning with a heading line followed by one or more sections, each consisting of one or more paragraphs of information).

While alternative A involves only a small set of code changes, there are several issues in the current help_ code that could be resolved by alternative B.

help_ & help_rql_: Issues with Current Code

The current help_ code is quite complex, making it difficult to understand and maintain. Several factors lead to the complexity.

- Use of a highly-flexible but non-standard "growing segment" mechanism to manage stored variables. This mechanism starts with input data provided to help_$init; continues with structures describing parsed components from the info segment; and ends with data returned by the help_$check_info_segs entry point.
    - Because it is non-standard, the "growing segment" mechanism isn't as well documented as more standard allocation schemes. Therefore, fewer programmers understand it.
    - This has resulted in coding errors in use of the mechanism, mostly in code inside help_ and help_rql_.
    - These coding errors produce unexpected and quite difficult-to-understand flaws in displaying info segments as earlier stored data is unexpectedly overlaid by later data. Attempts to reuse the earlier data then cause very wrong (sometimes only slightly wrong) data to be displayed.

- Delayed parsing of info seg contents: parse text only when needed for display. This delayed approach has several consequences.
    - Info seg parsing code is mixed with code to analyze and display info seg content, making both code components more complex.
    - Info seg parsing code is spread across help_ and help_rql_ code making it more difficult to understand and maintain.
    - Parsing code spread among many internal procedures forces some duplicated parsing code fragments.
    - Distributed parsing makes status of parsing operations more obscure: harder to tell whether a given part of the info segment has been parsed, or where structures generated by that parse reside in storage.
    - Adding help features to consolidate or summarize info data can be done only with difficulty. Examples include:

      help -brief
      help -control_arg STRING

Goals for help & help_

Any project to replace or revise the existing online help facility should be designed to achieve the following goals.

- Resolve the issues discussed above.
    - Use the new info_seg_ subsystem to pre-parse all info segment content before attempting to analyze or display that content.
    - Simplify or replace the "growing segment" storage mechanism to avoid current and future coding errors in help_.
- Modernize help code.
    - Make best use of today's high-speed communications mechanisms and user interface technologies in presenting the desired help data.
        - Wider and longer terminal screens and higher-speed communication channels allow more data to be returned between user prompts.
        - Video system screen content editing allows user prompts and responses to be removed from the terminal scroll-back buffers, leaving the info seg content displayed in a more readable form.
    - Use ssu_ mechanisms and interfaces for structuring and simplifying help subsystem code.

However, the existing online help facility is an integral part of the Multics system. Its user interface is well-documented and well-known to current Multics users. Its help and help_ interfaces are called by many Multics subsystems.

- The well-documented user interface to the help command should be changed as little as possible.
    - No existing features should be removed.
    - Additions should be minor in nature, especially any that change the documented user interface.
- The help$ssu_help_request interface should not be changed.
    - That interface is called by ssu_info_mgr_, and changes to ssu_ routines are not envisioned as part of the design described in this technical bulletin.
- The documented help_ subroutine interface should be changed as little as possible.
    - Version number of the help_args structure (in help_args_.incl.pl1) should not change.
    - No data items may be removed from the help_args structure. New elements could be added to the structure if they replace existing pad elements and do not change the layout or storage space used for elements of the structure.
    - The documented sequence for calling help_ entry points should not change.
        - Call help_$init to initialize the help_args structure.
        - Fill-in elements of help_args structure (path names, control arguments, etc.).
        - Call help_ to display one or more info segment blocks.
        - Possibly change help_args structure elements, and call help_ to display different info segments.
        - Call help_$term to cleanup storage associated with help_args structure.
- No help or help_ changes should require info segment content to change except as described in MTB-1004.

Strategies for help_

In revising the help_ code, the follow strategies will be used.

1) Use info_seg_ to initiate, parse contents of, and terminate selected info segments.
2) Use get_line_length_ and get_page_length_ to determine screen-size characteristics of the user's terminal. When displaying an info block, aggregate adjacent sections that will fit on the terminal page.
3) Expand content of help_ prompts to describe all sections available in the next paragraphs to be displayed.
4) Treat responses to help_'s "More help?" prompt as requests from the user. Use Subsystem Utilities (ssu_) to implement response routines as ssu_ requests.
5) Since different responses are accepted for prompts about subroutines versus non-subroutine info blocks, use a different ssu_ request table for subroutine and non-subroutine info block kinds.

6) Call help response routines to implement all help_args.Sctl control argument switches:
    help -brief
    help -control_arg STRING
    help -titles
    help -list_entry_points
    help -section TITLE_WORD -search STRING

7) Use a shared data structure called **help_info** as the standard ssu_ info_ptr argument provided to all ssu_ request procedures. This help_info structure consolidates global data for:
    a. tracking all info paths found by a help_ call;
    b. locating parsing results for each info segment;
    c. tracking status of the current info block display operation;
        i. recording current position within each info block being displayed;
        ii. selecting next range of sections to be displayed from the info block;
    d. tracking storage location of data needed by all help_ response routines.

8) Use the ssu_$listen loop to drive interactions with the user when displaying an info block.

help Command Changes

The help command provides the user interface for the help facility at Multics command level, and for the help request provided by ssu_info_mgr_ to most ssu_-implemented subsystems. Only minor, upward-compatible changes will be made to this user interface.

1) Add a new control argument permitting the help command to test locating subsystem info topics which use the new :[Info]: (hidden info block) divider. Syntax for this new control argument is:
   -Info HELP_INFO_NAME, -info HELP_INFO_NAME
   Such blocks appear only in subsystem info segments in which the subsystem's built-in help request searches for an info block in a known info segment. For example, probe.info contains a divider:
   ```
   :[Info]: stack: sk:   1985-05-04  stack, sk
   ```
   that might be accessed by the help command with this new -info control:
   help probe -info stack

   Before this change, help could locate an info block starting with an :Info: divider, because the names on that divider line were also external names on the info segment (with .info suffix added to each name). But for names on an :[Info]: divider line, those names are never used as external names on the info segment. This change allows such info blocks to be accessed by the help command for testing purposes. The subsystem's help request will access such hidden topic blocks without using -info.

2) Allow -case_sensitive (-cs) to be given with the control:
   -control_arg STRING
   Before this change, -case_sensitive could only be given with -search and -section controls. But the new help_ implementation applies -cs to -section, -search, and/or -control_arg operands.

3) Allow other control arguments to be given along with the -brief and -control_arg controls.

4) Add hidden (undocumented) control arguments useful in testing help_:
   -no_video, -nv
   -debug OPERAND, -db OPERAND

5) Reorder internal arrays holding names of help's control arguments so the new -info, -no_video, and -debug controls can have alternate names (-Info, -nv, and -db).

The -info control is documented as follows:

```
help help -ca -info

>user_dir_dir>Multics>GDixon>w>vi_07>help.info   (215 lines in info)

2020-10-23  help

Control arguments (selecting info segments):
-info BLOCK_NAME
   within an info seg selected by INFO_NAME, actually select a block
   designated by :[Info]: BLOCK_NAME: where BLOCK_NAME.info does not
   appear as an entryname on the info segment.  This is useful for
   displaying a subsystem’s request descriptions.
```

The -no_video and -debug controls are debugging arguments. They are documented only in the “hidden_args” info block of help.info.

```
help -pn help -info hidden_args -bfhe

   (26 lines in info)

2020-12-31  help's hidden control arguments

The following control arguments were added to the help command to
support debugging of the help_ subroutine.  These descriptions are
hidden to avoid presenting debug-related descriptions to general user
community.

List of hidden control arguments:
-no_video, -nv
   disables help_ code that overwrites prompt and user response after
   user replies "yes" to the prompt.

-debug OP
   turns on special debugging code in help_.  OP may be an integer
   between 1 and 14 inclusive.  Only values 1, 2, 3 were defined as of
   this writing.
     OP=1: enable debug_prompt and debug_print calls in
           help_listen_util_.  These show factors affecting the next
           prompt, and next text printed.
     OP=2: enable print statements in help_util_$format_LIST showing
           how number of columns and rows were determined in the
           formatted LIST.
     OP=3: print iFile.caseI and .structure values seen by help_ after
           info segment has been parsed.
   The OP value is stored in the help_args.pad2(6) array element, and
   help_ accesses that OP using that element.
```

help_ Subroutine Changes

The total rewrite of the help_.pl1 source preserves many aspects of the existing help_.pl1 module.

- The revision: preserves the documented entry point names and calling sequences for all help_ subroutine entry points; and adds no new help_ entry points.

help_: Use of ssu_

The existing help_ subroutine was written before the ssu_ subsystem was created, so its help_rql_ user interface was fashioned as a "helpful program" sequentially displaying information named by the user, prompting with title of each upcoming section, and asking if the user wants to see that information (the "More help?" prompt). The user then responds "yes" to see the next section, or gives another response to select a different section, jump to the next selected info block, or stop the display.

This simple user interface worked well when there are only a few help responses, each 1-word long. But info segment content has broadened. Help files initially described only commands and general info topics. Now they have evolved to document: commands that work as active functions; subroutines with many entry points; subsystems with many requests and active requests; multi-operation commands; I/O modules; etc. With each broadening of info seg content and format, the number and complexity of responses has increased.

More than 20 possible responses are now supported for each "More help?" prompt; and most responses accept one or more arguments or control arguments. Also, responses accepted when displaying a subroutine info seg (e.g., responses dealing with listing or selecting subroutine entry points) differ from those allowed when displaying a non-subroutine info segment. help_ needs to assist the user about responses that may be given for the current kind of info block, and the arguments they accept.

Services provided by ssu_ can simplify coding of help response routines. Also, the ssu_ listener loop can replace much of the code in help_rql_ that shepherds display of info sections. Also, ssu_ services may be tailored: the request table may be easily changed; and special operations may be performed before and after each request line is read and evaluated.

The following subsections describe how ssu_ is configured to display particular kinds of info segments.

help_$init Entry Point Changes

The existing help_$init performs the following steps:

- call get_temp_segment_ to obtain a temporary segment;
- overlay a **help_args structure** at the beginning of that segment;
- initialize help_args elements as if no help arguments or control arguments were given;
- store "info segment" (or a caller-specified group of) search paths in help_args.search_dirs.

The revised subroutine performs several more steps in setting up the operating environment for the help_ entry point. These include:

- initializing storage for a new **help_info structure** consolidating details about the entire help_ subsystem in one place;
- creating an **ssu_ subsystem** used exclusively by help_ and its supporting routines;
- creating the **info_seg_data structure**, which is the top-level structure of the *info segment structure hierarchy* used by the info_seg_ subsystem in parsing info segments.

With this revision, the temporary segment used by the help_ interfaces is supplied by ssu_. And this temp seg will contain the three structures mentioned above when help_$init returns to the caller. Order of these structures is:

1. **help_info**
2. **info_seg_data**
3. **help_args**

Only the help_args structure is visible to the caller, and it must be the last item allocated in the temporary segment, so the caller may include its flexible-extent array elements. The help_info structure is used exclusively by the help_ entry point, its supporting ssu_ requests ("More help?" response routines), and the help_ supporting utilities. The info_seg_data structure is part of the data interface between help_ and the info_seg_ subsystem.

Steps performed by the revised help_$init are shown below.

- Setup a cleanup on-unit to dispose of the ssu_ invocation if initialization fails or is interrupted.
- Create the ssu_ subsystem which is used exclusively by help_ and its supporting routines. It starts out with:
    - no request table
    - no info directory
    - an info_ptr (to structure communicating between help_ and its requests) set to null.
- Call ssu_$get_temp_segment to obtain the temp seg for the three structures named above.
- Overlay a **help_info** (**hi**) structure as first item in that temporary segment. This structure is declared in _help_shared_data_.incl.pl1.
    - Call ssu_$set_info_ptr with addr(help_info) as its argument. ssu_ then passes a pointer to the help_info structure to all help_ requests; and it is available to any help_ support routines that receive an sci_ptr parameter (via ssu_$get_info_ptr).
    - Initialize elements of the help_info structure. A major element involves the next_free_spaceP pointer, which is intended to point to the next free doubleword in the temp seg.
    - Perform a *delayed-allocation* of the help_info structure, once it has been initialized. [For details, see the subsection below "help_info - Delayed-Allocation Storage Mechanism in the Temporary Segment".]

- Overlay an **info_seg_data** (**isd**) structure as the second item in the temporary segment.
  - hi.isdP is the locator variable for this structure. (Remember that hi is the short name for the help_info structure.)
  - Initialize the structure so the help_$help_ entry point can use it for calls to the info_seg_ subsystem.
  - Perform a *delayed-allocation* of the isd structure in the temp seg.

- Overlay the **help_args** structure as third item in the temporary segment.
  - The hi.help_argsP is the locator variable for this structure. The Phelp_args parameter of help_, help_$init and help_$term entry points also points to this structure.
  - Initialize help_args with values identical to those provided by the existing help_$init. (The help_args structure used by callers of help_ is unchanged.)
  - Perform a *delayed-allocation* of the help_args – as initialized here. However, the caller of help_$init usually extends its array elements which increases size of this structure when the help_$help_ entry point is called. [Note: help_$help_ will redo the delayed-allocation of help_args to account for this enlargement of the help_args data.]

help_info (hi): A New Structure

An ssu_ subsystem passes subsystem-wide data from the main application to each request using a data structure pointed to by the info_ptr argument passed as an argument to each request.

For the help_ subsystem, the ssu_ info_ptr points to the help_info structure introduced in the section above. This section shows the declaration and describes elements of that structure. Note that the help_info structure is also declared with a shorter alias name (hi) so its elements may be referenced with a more compact name (e.g., hi.info_ptrs or hi.help_argsP).

The help_info structure is the first storage residing in the temporary segment help_$init obtains from ssu_.

help_info - Delayed-Allocation Storage Mechanism in the Temporary Segment

The current "growing segment" mechanism for allocating storage in a raw Multics segment has many draw-backs. However, it does offer a major benefit. The structure residing at the end of the segment can grow or shrink in size simply by changing structure elements referenced in each <refer option> that define the current extent for character strings or array dimensions declared in the structure. The storage only gets officially "delay-allocated" (recorded as used space in the temp seg) after the final setting of these adjustable extents.

help_ uses many adjustable-extent structures, but only builds them one-at-a-time. These include:

- **help_args** (variable number of info_segment search paths, info seg pathnames, operands for -section and -search control arguments, etc.).
- help_ builds a "list of info seg pathnames" as it evaluates input paths, expands them by searching using search paths and starnames. help_ expands that list a second time into a **"selected info blocks"** array for possible display by the user.

- help_ constructs lists of: subroutine entry points; of items in “Arguments”, “Control arguments”, and “List of …” sections; help_ response names; etc.

The raw-segment storage is perfect for such uses because it allows adjustable-extent structures to grow incrementally; and to be released using a single assignment operation.

The **delayed-allocation** mechanism is managed using:

- **hi.next_free_spaceP,** a pointer to the next unused doubleword-aligned storage in the temporary segment; and
- **hi.info_ptrs.storage**, pointers to each of the structures that might be delay-allocated in the temporary segment. These pointers: locate structures already allocated; and provide a mechanism for releasing storage at a given location, along with other structures stored from that location to end of the segment.

Storage for a based structure declaration is *delay-allocated* by:

A. **assigning hi.next_free_spaceP** to the pointer locating the structure being allocated;
B. setting any adjustable extents in elements of that structure until it has reached full-size; then
C. **calling set_space_used** with the current size of that structure, which adjusts hi.next_free_spaceP to point to the doubleword just beyond the end of the structure[1]:

```
call set_space_used ( currentsize(based_structure) );
```

RULE: Only structure(s) just prior to the hi.next_free_spaceP pointer may be released. This means that storage for a based structure may be released if its locator value is known, and the locator values for all structures possibly allocated after that structure are also known.

From a practical standpoint, this means storage must be delay-allocated in a known order. If the allocation order for several structures is not predictable, they may be treated as a *delay-allocation group* stored in adjacent locations at the end of the temporary segment. Structures in the group can be delay-allocated in any order. But once one group member is allocated, structures not belonging to this group may not be allocated until the entire group is released (removed) from the temporary segment. Furthermore, the entire storage for all group members must be released at the same time.

---

[1] The PL/1 currentsize(X) builtin function returns the number of words occupied by the generation of storage obtained by evaluating the reference X. If X is a based variable with <refer option>s, the result depends on the <reference> contained in the <refer option>, giving size of the variable after adjustable extents stored in that structure have been set.

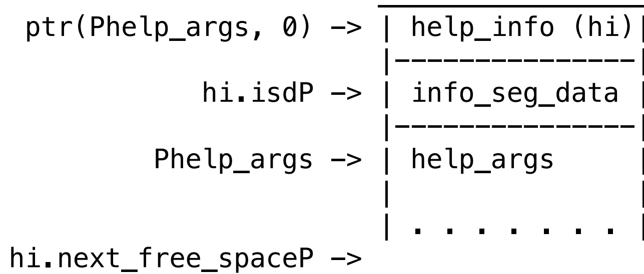
Compare this to the size(X) builtin function which returns a word count that depends on the <expression> given in the <extent expression>. For a delayed-allocation variable, such <expression> is usually set to 0 since such variables are never referenced in a regular <allocate statement>.

For example, in the declaration for STRING shown below, the STRING.text element is declared as character data using an adjustable extent with 0 value in its <extent expression> and a <refer option> referencing its STRING.L element.

```
dcl 1 STRING aligned based(STRING_P),
      2 L fixed bin(21),
      2 text char(0 refer (STRING.L)),
       STRING_P ptr;
```

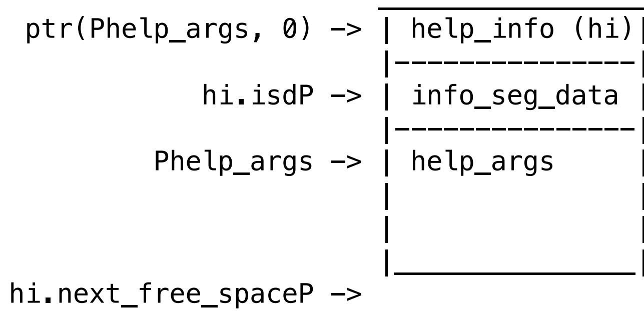
help_$init always returns with three structures in the temp segment. help_args is the last structure. The caller of help_$init can change its adjustable-extents to grow or shrink the structure.

Figure 1: Temp Segment after calling help_$init



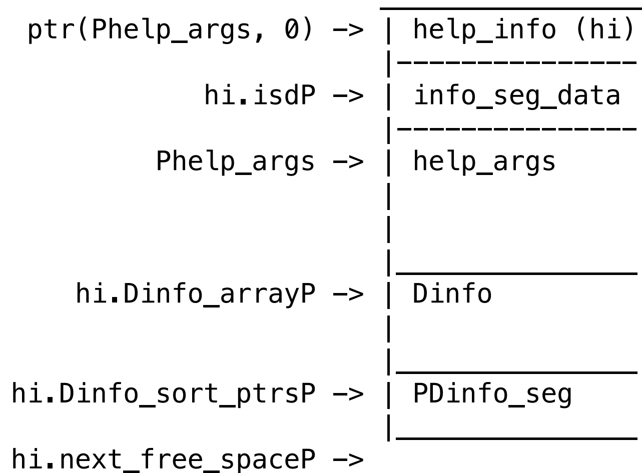
Each time help_$help_ is called, its caller has put new data in help_args and set all of its adjustable-extent elements. Thus, help_$help_ re-performs the *delayed-allocation* of the help_args structure to record its current size. [Note: Data within the help_args structure is not affected by this redo of the delayed-allocation. Only the hi.next_free_spaceP pointer value is adjusted by the call to set_space_used.]

Figure 2: Temp Segment after help_$help_ progress = 3



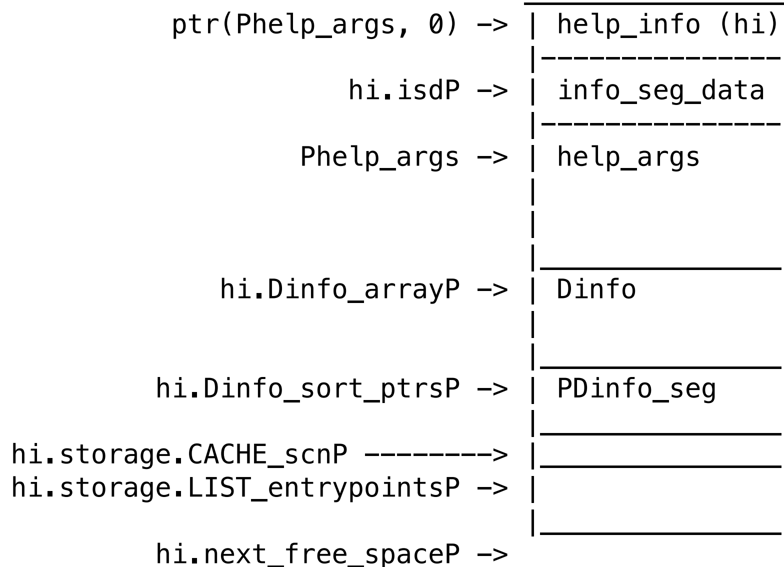
help_$help_ then incrementally builds two additional structures, delay-allocating them in a fixed order in the temp segment. The Dinfo structure holds an array of Dinfo_seg structures, each identifying one info block to display. The PDinfo_seg structure holds an array of pointers to each of the Dinfo_seg structures in the Dinfo array. These pointers are sorted into the order in which help_ will display those info blocks. After sorting, the layout of structures in the temp segment are as follows.

Figure 3: Temp Segment before help_$help_ calls display_block



help_$help_ calls its display_block procedure to display each of the info blocks in sorted order. display_block and the help_responses_$XXX routines and help_util_ subroutines perform *delayed-allocations* in the temp seg as part of displaying each block. These allocations can occur in many different sequences depending upon the kind of info block, and the user's "More help?" responses. Locator values for delay-allocated structures are all kept in elements of the hi.storage substructure.

Figure 4: Temp Segment when running display_block



Each time display_block is called to display a new info block, it begins by setting all of the hi.storage elements to null pointers, thus releasing any items delay-allocated by earlier calls. And each time display_block returns to help_$help_, that routine adjusts hi.next_free_spaceP back to the position shown in Figure 3 above. This pair of operations ensures that fresh structures describing items in each new info block get delay-allocated during each call to display_block.

Storage is released to the state shown in Figure 3 as follows:

A. help_$help (NEXT_INFO label):
    - sets: hi.next_free_spaceP = hi.Dinfo_sort_ptrsP;
    - calls set_space_used (current_size(PDinfo_seg));
B. display_block nulls all storage allocations from displaying any prior block:
    - sets: hi.storage = null();

Before help_$help returns to its caller, it releases storage it allocated in Figure 3, thus returning the temp segment to the state shown in Figure 2 (exactly as when its caller invoked the help_ entry point).

A. help_$help (SUBSYSTEM_ABORTED label):
    - sets: hi.next_free_spaceP = hi.help_argsP;
    - calls set_space_used (current_size(help_args));

Note that some routines called by display_block use this *delayed-allocation* storage space as a temporary scratchpad. They assign the value of help_info.next_free_spaceP to a based structure locator, fill-in and use that storage in performing the given response, but never call the **set_space_used** procedure to allocate that storage. Since .next_free_spaceP is unchanged when that response routine returns, storage used for its scratchpad structure is overwritten when the next program gets space at the end of the temporary segment. Such scratchpad use is permitted even when members of the delayed-allocation group have been allocated.

help_info - Operating Environment

The first part of the help_info structure declares pointers to elements of help_'s working environment. These help_info.init_ptrs are setup by help_$init. They include:

- the ssu_ subsystem invocation data pointer for help_;
- a pointer to a standard allocation area;
- a pointer to the info_seg_ invocation data;
- a pointer to the help_args structure; and
- the pointer to the next free doubleword in *delayed-allocation* storage.

Storage for the info_seg_ subroutine invocation data and help_args structures are allocated in the temporary segment as the second and third items. (The first item is the help_info structure itself.)

```
dcl 1 help_info aligned,          /*     .. Position 1 (base) of help_ temp segment.        */

    2 init_ptrs,                  /* == Items set by help_$init call.                    == */
      3 (sciP,                    /*  - help_ subsystem's ssu_ invocation info_ptr          */
         areaP,                   /*  - Points to help_ subsystem's standard allocation     */
                                  /*    area (supplied by ssu_).                            */

         isdP,                    /*  - Points to info_seg_ subsystem's info_seg_data struct*/
                                  /*     .. Position 2    within help_ temp segment.        */

         help_argsP,              /*  - Points to help_args (help_ input arguments from     */
                                  /*    help command or caller)                             */
                                  /*     .. Position 3    within help_ temp segment.        */

         next_free_spaceP         /*  - Points to next available space in help_ temp seg.   */
         ) ptr,
```

help_info - Data for Found Info Blocks

The second part of the help_info structure declares pointers to the result of help_$help searching for info segments and blocks named in help_args.path(*) elements. They include:

- an array whose elements describe info blocks selected, and the info segment that contains that block.
- an array of pointers to elements of the first array. Those pointers can be sorted for uniqueness of the matching entry name on the info block and its containing info segment.
- a pointer to the element of the first array which is being displayed by help_.

```
...
    2 help_ptrs,                   /* == Items set by help_$help_ call.               == */
      3 (Dinfo_arrayP,             /*  - Points to Dinfo structure containing an array of */
                                   /*    Dinfo_seg structures, each describing one info   */
                                   /*    block matching a help_args.path(I) input argument. */
                                   /*     .. Position 4    within help_ temp segment.       */

         Dinfo_sort_ptrsP,         /*  - Points to PDinfo_seg structure containing a sorted */
                                   /*    array of pointers to one of Dinfo_seg structures.  */
                                   /*     .. Position 5    within help_ temp segment.       */

         DinfoP                    /*  - Points to current Dinfo_seg structure.            */
         ) ptr,
```

help_info_ Data for Displaying Info Blocks

The remaining parts of the help_info structure are used by the display_block internal procedure of help_. This procedure is responsible for all aspects of displaying the current info block. The procedure is described in a later section of this bulletin. Its shared data includes: user terminal information; interface information between help_ and the display_block procedure; storage pointers for response routines and display_block; and display_block state data.

The third part of the help_info structure declares information about the user's terminal. If the terminal is connected using the window_io_ I/O module (the video system), display_block tries to remove prompt lines and the user response from the terminal's scroll-back buffer when printing a sequence of info segment sections. Only sections of the info block remain in the scroll-back buffer.

```
...
    2 help_video_data,               /* == Items set by help_$help_ & its display_block rtn == */
      3 video_iocbP ptr,             /*  - Points to IOCB of process's active video terminal.  */
      3 prompt_region,               /*  - Upper-left line/col index and lines/cols extent of  */
        4 (lineI, colI, linesN, colsN) fixed bin,
                                     /*    region containing most recent prompt & user response*/
      3 clear_prompt_regionS bit(1) aligned,
                                     /*  - if T: clear prompt region before displaying pghs.   */
      3 hvd_pad bit(36) aligned,
```

The fourth part of the help_info structure provides label variables permitting display_block and response routines to perform a non-local transfer to particular areas of code within help_ when a particular event occurs.

...

```
    2 help_labels,                 /* == Labels set by help_$help_, used by display_block == */
                                   /*    help_listen_util_ & help_responses_               */
                                   /*    to handle asynchronous events.                    */
                                   /*  Transfer to given label when response ...           */
      3                    NEXT_INFO_LABEL                   label                  variable,
                                   /*  - requires skipping to next file.                   */
      3 PI_LABEL label variable,   /*  - interrupted via program_interrupt signal.         */
      3                   ALL_PARAGRAPHS_LABEL                label                 variable,
                                   /*  - wishes to print all paragraphs of current block,  */
                                   /*    or all blocks of a subroutine info seg.           */
      3                   ANOTHER_BLOCK_LABEL                 label                 variable,
                                   /*  - requires changing to another block in same info seg.*/
      3                  SUBSYSTEM_ABORT_LABEL                label                 variable,
                                   /*  - requires exiting help_ subsystem.                 */
```

The fifth part of the help_info structure provides a data interface between help_ and display_block, except for newline_Nblanks_output which tracks number of consecutive blank lines emitted by display_block and the response routines via the newline() shared procedure.

```
...
    2 help_numbers,                /* == Items set by help_$help_, used by header response== */
      3 (Dinfo_sort_ptrsI,         /*  - Index of current Dinfo_seg structure's pointer in   */
                                   /*    PDinfo_seg.P array of pointers to Dinfo_seg structs */
        Dinfo_last_cross_refI,     /*  - Index of last Dinfo_seg structure's pointer in the  */
                                   /*    PDinfo_seg.P array of pointers w/ Scross_ref = T.   */

                                   /* == Items set by help_$help_ and display_block       == */
        infos_printedN,            /*  - Count of Dinfo_seg info blocks displayed by help_.  */

                                   /* == Items set by help_$help_, display_block,         == */
                                   /*    help_listen_util_, help_responses_, etc.            */
        terminal_lineL,            /*  - Max length for lines on current terminal.           */
        newline_Nblanks_output,    /*  - Count of blank lines emitted w/o additional output. */
        hn_pad                     /*  - Pad field for alignment of pointers that follow.    */
        ) fixed bin,
```

The sixth part of the help_info structure locates storage allocated by display_block and response routines in help_'s temporary segment.

```
...
    2 info_ptrs,                   /* == Data set/used by help_'s display_block and      ==  */
                                   /*    help_responses for each block.                      */
      3 storage,
        4 (CACHE_caP,              /*  - Points to cache of default control_arg resp args.   */
          CACHE_findP,             /*  - Points to cache of default find_info response args. */
          CACHE_scnP,              /*  - Points to cache of default section response args.   */
          CACHE_srhP,              /*  - Points to cache of default search response args.    */

          STRING_entrypointsP,     /*  - Points to STRING w/ list of subr entrypoints        */
          LIST_entrypointsP,       /*  - Points to LIST of subr entrypoints in current info  */
          LIST_non_subroutine_responsesP,
                                   /*  - Points to LIST of help non-subroutine responses     */
          LIST_subroutine_responsesP,
                                   /*  - Points to LIST of help subroutine responses         */
          LIST_helpRql_responsesP  /*  - Points to List of help request loop responses       */
          ) ptr,                   /*     (not currently used since help request does not    */
                                   /*      prompt user and accepts no responses)             */
```

The seventh part of the help_info structure contains pointers to: the current iFile and iBlok being displayed, the "current paragraph" to be displayed next; and the suggested range of paragraphs that will be displayed. The iFileP and iBlokP pointers are set by display_block when it is called by help_. The current paragraph information is adjusted by various response routines and paragraph display routines. iPgh_print_range is set by help_listen_util_$display_prompt and help_listen_util_$set_iPgh_range.

```
...
                                    /* == Data for the "current block" being displayed.    == */
      3 (iFileP,                    /*  - Points to its iFile structure.                      */
         iBlokP                     /*  - Points to selected block within iFile.              */
         ) ptr,

      3 selected,                   /* == Data for tracking "current paragraph" of block.  == */
        4 (iSectP,                  /*  - Points to selected section; null at "top" of block. */
           iPghP                    /*  - Points to selected paragraph; null to select sect.  */
           ) ptr,

      3 iPgh_print_range,           /* == Data given next chunk put onto terminal page.    == */
        4 (startP,                  /*  - Points to 1st paragraph to be displayed on cur page */
           endP                     /*  - Points to last pgh to be displayed on current page  */
           ) ptr,                   /*    of display_block output.                            */
```

The eighth part of the help_info structure contains state information shared by display_block and the response routines when displaying an info block, or navigating through sections of that block.

```
...
    2 info_numbers,                 /* == Data set by help_'s display_block procedure.     == */
      3 (block_progress,            /*  - Progress selecting/searching/displaying info block. */
                                    /*     (see BLOCK_PROGRESS_xxx constants below)           */
         display_mode,              /*  - Current block is non-subroutine vs. subroutine.     */
                                    /*     (see DISPLAY_MODE_xxx constants below)             */
         display_limit,             /*  - Limits on output paragraphs.                        */
                                    /*     (see DISPLAY_LIMIT_xxx constants below)            */
         header_Nlines_follow       /*  - Count of lines to be displayed immediately after    */
                                    /*    current help_responses_$header output.  This value  */
                                    /*    is zeroed when header has been displayed.           */
         ) fixed bin,

    2 info_switches,                /* == Switches controlling display/prompt content.     == */
      3 (print_inhibitS,            /*  - If true, inhibit any printed output.                */
                                    /*        Set at PI_LABEL targets.                        */
                                    /*        Cleared when ready to display next prompt.      */
         prompt_repeatS,            /*  - If true, inhibits display of next info paragraphs,  */
                                    /*    and forces prior "More help?" prompt to be repeated.*/
                                    /*        Set when response diagnoses user error, or      */
                                    /*        non-pgh request returns.                        */
         section_search_matchedS,   /*  - Result returned by most recent section/search/find  */
                                    /*    response.                                           */
         another_blockS             /*  - If true, selects header format w/o path of info seg.*/
                                    /*        Set when switching to another info block after  */
                                    /*        earlier printing of full header.  It's same     */
                                    /*        info seg, just a different block.               */
         )  bit(1) aligned,
      3 Sctl aligned like help_args.Sctl;
                                    /* Copy of help_args.Sctl switches which can be changed   */
                                    /*   by display_block or responses as display ops occur.  */
```

The following declaration is a naming shortcut for the help_info structure. Code refers to its elements as hi.XXX.

```
dcl 1 hi aligned like help_info based (help_infoP),
                                    /* Short name for help_info structure.                    */
     help_infoP ptr;
```

The following declarations provide constants for possible values of the hi.block_progress, hi.display_limit, and hi.display_mode elements.

```
dcl (BLOCK_PROGRESS_new_block         init(1),
                                    /* Starting to process a new info block (Dinfo.seg item). */
     BLOCK_PROGRESS_section_search    init(2),
                                    /* Searching within block for starting section/paragraph. */
     BLOCK_PROGRESS_needs_header      init(3),
                                    /* First header response emits full header & increments   */
     BLOCK_PROGRESS_display           init(4)
                                    /* Entering request loop to display info block sections.  */
     ) fixed bin int static options(constant);

dcl (DISPLAY_LIMIT_none               init(0),
                                    /*  - Set when displaying all paragraphs that fit on page.*/
     DISPLAY_LIMIT_section            init(1),
                                    /*  - Set when displaying only paragraphs of 1 section.   */
     DISPLAY_LIMIT_unseen             init(2),
                                    /*  - Set when displaying only unseen paragraphs.         */
     DISPLAY_LIMIT_rest_unseen        init(3)
                                    /*  - Set when displaying all remaining unseen paragraphs.*/
     ) fixed bin int static options(constant);

dcl (DISPLAY_MODE_unset               init(0),
     DISPLAY_MODE_non_subroutine      init(1),
                                    /*  - Set when displaying non-subroutine info block       */
     DISPLAY_MODE_subroutine          init(2),
                                    /*  - Set when displaying subroutine w/ entrypoints block */
     DISPLAY_MODE_help_request_loop   init(3)
                                    /*  - Set when invoking help from help_'s request loop    */
     ) fixed bin int static options(constant);
```

help_$help_ Entry Point Changes: progress = 1, 2

Code in the help_$help_ entry point tracks steps for completing each call by setting the progress parameter. For progress step 1, help_ validates the version number of its main input argument, the help_args structure.

- The declaration for the help_args structure (in help_args_.incl.pl1) is unchanged.
  - Names, attributes, and order of elements are the same.
  - Comments documenting each element were changed to better-describe purpose of each element.
  - The help_args.version element is still set to Vhelp_args_3.
  - The help_args.sci_ptr element is ignored.
  - The help_args.help_data_ptr element is ignored.

The help_args.sci_ptr element is ignored because help_'s new implementation would upset the caller's ssu_ environment. The new implementation: sets its own ssu_ request tables; changes several ssu_ $listen replaceable procedures; and passes its ssu_ invocation to the info_seg_ routine for its own use. These new behaviors would disrupt any ssu_ environment passed in via the help_args.sci_ptr element.

The existing help_ implementation sets help_args.help_data_ptr to point to a second ssu_ invocation that it creates to implement its own `help` response. The `help` response in the new implementation displays info blocks describing help_'s responses without need for this second ssu_ invocation.

Progress step 2 checks to see that 1 or more help_args.path(*) substructures provide names for info segments to be displayed.

help_$help_ Entry Point Changes: progress = 3

The help_ entry point code for progress step 3 accesses storage for the new help_info structure, located at the beginning of the temporary segment holding the help_args structure. The help_info.init_ptrs elements were initialized by help_$init; most sub-elements are not changed by help_$help_. Other level 2 elements of this structure are re-initialized for this invocation of help_$help_.

The help_info.init_ptrs.next_free_spaceP points to the first doubleword of free space in the help_args temp segment following the current help_args structure. Since help_args will change in size/content for each call to help_, help_info.next_free_spaceP is set to location of the start of help_args. Then the set_space_used procedure is called to adjust the next_free_spaceP to the doubleword following current contents of the help_args structure. The existing help_ had similar code. The new implementation simplifies and standardizes all instances which use help_info.next_free_spaceP and set_space_used to manage this temporary storage.

The final action in progress step 3 calls the evaluate_path procedure to examine/expand/evaluate each of the help_args.path(N) substructures. Any errors found are reported, and a non-zero status code is returned to the caller of help_. evaluate_path follows the same algorithm used in the existing help_.pl1 routine.

help_$help_ Entry Point Changes: progress = 4

The help_ entry point code for progress step 4 gets information about the user's terminal line length and lines per page. help_ uses this information to suggest up to a page-full of info sections in each "More help?" prompt.

If the terminal is using the window_io_ video system, help_ notes that attachment. Then prompts and user responses can be removed (overwritten on the screen) as display of consecutive info sections proceeds. This leaves only the paragraphs of the info block in the terminal's scroll-back buffer. The help_args.Sctl.no_video switch (help -no_video hidden control) suppresses such removals as an aid to debugging help_'s prompts and user response processing.

A new info_seg_$init_for_help_ entry point is called to quickly initialize the info_seg_ subsystem for parsing info segments.

help_$help_ get_info_seg_list Procedure

The final code in progress step 4 calls the get_info_seg_list procedure to map each help_args.path(J) substructure into one or more info segment pathnames.

The revised code still builds a list of info segment paths by:

- searching for INFO_SEG_NAME.info using the "info_segment" search paths; or
- checking the existence of segment at: help_args.path(J).dir || ">" || help_args.path(J).ent (where the ".info" suffix has been appended if missing from the .ent element).

Each help_args.path substructure can identify several info segments if the star convention is used in path.ent or path.info_name elements.

If the .path substructure includes no specific directory, code searches for a segment using the info_segments (or other caller-specified) search paths. An info segment matching the given path.ent may be found in several of the search directories. The segment found in the topmost search directory is the **primary info segment** to be displayed. Other elements with the same entry name in lower search directories are called **cross reference segments**. Their Dinfo_seg.Scross_ref switch is set to TRUE. (See the Dinfo_seg declaration below.)

This code was cleaned up (re-indented in the style used for the revised help_ subroutine) and somewhat simplified. Most of the algorithms used in this revised code are identical to those of the existing help_ subroutine.

Code in get_info_seg_list that parsed an info segment into blocks was replaced with a call to the new info_seg_$examine_iFile entry point. This entry point performs only the initial work of info_seg_$parse_iFile: parsing the info segment into lines and blocks, and determining its file structure. get_info_seg_list then adds items to the *selected info blocks* array.

An item in the array has the following declaration (from help_cis_args_.incl.pl1).

```
dcl 1 Dinfo_seg    aligned based, /* Information  about one info segment.          */
                                  /* NOTE: code depends upon Scross_ref and ent     */
                                  /*   staying 1st & 2nd elements of item substruc. */
    3 Scross_ref   bit(36) aligned,
                                  /*   bit 1 on if same info appears in diff. dirs. */
    3 ent          char(32) unal, /*   ent part of info block pathname.             */
    3 info_name    char(32) unal, /*   info_name used to find info, if different    */
                                  /*     from ent (without suffix).                 */
    3 dir          char(168) unal,/*   dir part of info block pathname.             */
    3 ep           char(32) var,  /*   subr entrypoint name requested in pathname.  */
    3 uid          bit(36),       /*   unique ID of containing segment.             */
    3 I            fixed bin(35), /*   index of 1st character of the info.          */
    3 E            char(32),      /*   ep name to be used for sorting               */
    3 X            fixed bin(35), /*   index of original record order               */
    3 pad2         fixed bin(35),
    3 L            fixed bin,     /*   length, in chars.                            */
    3 date         fixed bin(71), /*   date_time_entry_modified of info segment.    */
    3 (segment_type
                   fixed bin(2) uns,
                                  /*   00 - link, 01 - segment                      */
     mode          bit(3),        /*   access mode.                                 */
     pad1          bit(31)
     )             unaligned,
    3 code         fixed bin(35); /*   error code encounter in processing seg.      */
```

A particular block is selected from each item in the array, as follows:

- If the info seg has no block dividers, the entire segment is treated as an info block selected for possible display.
- For info segments having block divider lines:
  - If the info seg ends with a history comment block, that history comment iBlok structure is removed from the threaded list of blocks returned by info_seg_$examine_iFile. History comments are ignored by help_.
  - If any info blocks contain :Entry: dividers, the entire subroutine info segment is selected for possible display. Selection of a particular info block is deferred to the display_block procedure.
  - If the info blocks contain only :Info: or :[Info]: dividers, code looks for a block whose name matches either: the help_args.path(J) specification to ensure that the name is also an external name on the info segment; or if help_args.path(J) includes a separate .info_name value, a block name matching the separate .info_name value. If a block matches one of these name constraints, that info block is selected for display.

This examination follows the algorithm used by the existing help_ get_info_seg_list subroutine, but does the parsing and final examination of the info segment using data provided by info_seg_$examine_iFile.

help_$help_ Entry Point Changes: progress = 5

The help_ entry point code for progress step 5 sorts the "selected info blocks" array by info seg entry name and block name, eliminating any duplicates from the list. This code uses the same algorithms as the existing help_.

help_$help_ Replacement for help_rql_

At this point, the existing help_$help_ code calls the help_rql_ subroutine to display each info block in the "selected info blocks" list, one section at a time. After displaying the first section, it asks the user whether the upcoming section should be displayed. help_rql_ does most of the parsing of the info segment, runs the display/prompt request loop, and contains code for each of the possible responses. As you might expect, help_rql_ is quite complex.

The new code uses a different strategy. First, help_$help sets up the ssu_ environment for displaying info segments.

- Calls ssu_$set_prompt_mode to DON'T_PROMPT. help_ issues its own "More help?" prompts.
- Calls ssu_$set_procedure to replace the pre_request_line, post_request_line, program_interrupt and unknown_request procedures with routines from help_. These are described below in the "help_ Changes for Incorporating ssu_" section.

For each item in the "selected info blocks" array, the new code does the following.

- Sets hi.PI_LABEL and hi.NEXT_INFO_LABEL to the NEXT_INFO label at the end of "selected info blocks" iteration loop.
    - That enables the user's `next` or `no` response to a "More help?" prompt to stop displaying the current block, and move on to the next info block in the iteration.
- Initializes the hi.info_ptrs substructure to null(). These point to the main data needed in displaying an info block.
- help_request_tables_.alm defines two ssu_ requests tables: one for subroutine info segments; a second for non-subroutine info segments. The new code calls ssu_$delete_request_table for the request table setup for the prior info segment.
- If more than 10 info segments have already been initiated and parsed, new code calls info_seg_$reinitialize to release storage holding the parsed info seg structures and terminate their info segments.
- Calls info_seg_$examine_iFile to set the **iFileP** variable to point to the iFile structure for the current info seg in the list, initiate that info segment, and parse it into lines and blocks.
- Calls info_seg_$parse_iFile to parse further parse each block into its component heading line, paragraphs and sections.
- If the info segment ends with a history comment block, unthreads that iBlok structure from its iFile.bloks thread. help_ never wants to examine history comments.

- Sets the **iBlokP** variable to point to the iBlok structure describing the block to be displayed, as follows:
  - If the info segment has no block divider lines (it contains only one un-named block), select that iBlok structure for display.
  - If the info segment describes subroutine entry points, select the first iBlok (subroutine introduction) for display; or if a particular entry point name was specified, search for the iBlok with block name matching that given entry point name. Report an error if no matching entry point block is found.
  - For non-subroutine info segment, select the iBlok found by get_info_seg_list (when segment was added to the "selected info blocks" array.
- Calls the new **display_block** internal procedure to display the selected block. More details about display_block are given in the next subsection. However, a few details are pertinent now to explain display_block's relationship with the earlier help_$help_ code.
  - display_block begins by again initializing all hi.info_ptrs elements to null values. It then copies the **iFileP** and **iBlokP** values into corresponding hi.info_ptrs elements. This makes those pointers available to each of the "More help?" response routines).
  - display_block may push storage for data related to the selected block onto the end of the temp segment holding the help_info and help_args structures. Pointers to these pushed data structures are kept in elements of the hi.info_ptrs.storage substructure.
  - display_block usually does not return directly to its caller. It calls ssu_$listen to interact with the user (give "More help?" prompts and display info seg text). ssu_$listen operates in an infinite loop. When all sections of the info block have been displayed, or the caller response indicates that display of the block should stop, the code does a non-local goto targeting the hi.NEXT_INFO_LABEL, which branches to the NEXT_INFO label.
- NEXT_INFO: is a label declared by help_ at the end of the "selected info blocks" iteration loop. When display of the selected block ends, code at this label releases display_block's data for that block from the help_info temporary segment. (hi.info_ptrs.storage pointing to that storage are nulled at the beginning of the next iteration through the "selects info segments" list.)
- The iteration loop repeats at this point, processing the next item in the "selected info blocks" array.

When all items in the "selected info blocks" array have been displayed, or the user responds with a `quit` request, help_$help_ does the following:

- Sets help_$help_ status code parameter to error_table_$nomatch if no info blocks were printed.
- Calls info_seg_$reinitialize to release all storage for info segment structures (iFile, iBlok, iSect, iPgh and iLine structures), and terminate any initiated info segment files.
- Adjusts hi.next_free_spaceP to point to the doubleword just beyond the end of the help_args structure, and truncate the containing temp segment to release all pages of that segment beyond that doubleword.
- Returns to the caller of help_$help_.

help_$help_ display_block Procedure

The existing help_ software calls a separate help_rql_ subroutine to display each block in the "selected info blocks" array. That routine has extremely complex code which:

- Processes control arguments given in help_args.Sctl.
- Parses each section of the info block as the decision is made to possibly display that section.
- Runs the loop which actually displays info block sections. This loop:
  - displays a "More help?" prompt describing the next section which will be displayed.
  - reads the user's response to the "More help?" prompt; parses that response and invokes a routine to implement each particular response.
  - displays either the next info block section, or whatever section was selected by the most recent user response.
  - when all sections of the current info block have been seen:
    - for a subroutine info block, switches to the next entry point block for possible display.
    - for a non-subroutine info block, returns to the help_ entry point which selects the next block for display (if any).
- Contains the internal routines that implement each possible user response.

The revised help_ software calls the **display_block** internal procedure to display each block in the "selected info blocks" array. display_block uses variables set by its caller to locate the info segment and block to be displayed, and therefore has no entry point parameters.

display_block does the following:

- Verifies that caller's iFileP and iBlokP are non-null; these two variables identify the info block to be displayed. It saves these variables in the help_info structure (hi.iFilep and hi.iBlokP).
- Resets other display_block-owned variables in help_info to known starting values. This releases storage delay-allocated by display_block, help_responses_ and help_util_ when displaying a prior info block.
- Determines whether the block being displayed contains subroutine or non-subroutine content; then:
  - adds an ssu_ request table appropriate to that content.[2]
  - sets hi.display_mode to either DISPLAY_MODE_subroutine or DISPLAY_MODE_non_subroutine.

- Sets hi.block_progress to BLOCK_PROGRESS_new_block.
- Copies the control arguments given in help_args.Sctl to hi.Sctl, so they can be tested by help response routines, and cleared as operations are performed for the block being displayed.

[2] Remember that the help_ loop walking through the "selected info blocks" array deleted the request table used for any previous block being displayed.

- Sets hi.block_progress to BLOCK_PROGRESS_section_search.
- If hi.Sctl.scn (help -section control) or hi.Sctl.srh (help -search control) is TRUE, runs tests on that block.
    - Calls help_util_$execute to run the "section" and/or "search" requests(s). (The help_$execute subroutine calls ssu_$execute_string to invoke a request line, and checks return code for occurrence of program_interrupt condition or call to ssu_$abort_subsystem while the request line was executing.)
    - Returns to the caller of display_block if the current block fails either of these tests.

- Sets hi.block_progress to BLOCK_PROGRESS_needs_header. Increments hi.infos_printedN to indicate that something is about to be displayed for the current block.
- If either hi.Sctl.he_only (help -header control) or Dinfo_item.Scross_ref is TRUE, then:
    - Calls help_util_$execute to run the "header" request to display either: the header; or a message noting that another info segment having the same name as the prior info block was found in a later search directory. (Only the info segment found in the earliest search directory is actually displayed.)
    - Returns to the caller of display_block.

- At this point, process any other help control arguments.
    - If hi.display_mode = DISPLAY_MODE_subroutine, then calls help_responses_$lep_setup to ensure there is an "Entry points in …" section at the end of the introductory block of the subroutine. Fabricate the list of entry points which is displayed in that section.
    - If hi.Sctl.all (help -all control) is TRUE, then positions to top of current block, and calls help_util_$execute to run the "header" request.
    - If hi.Sctl.bf (help -brief control) is TRUE, calls help_util_$execute to run the "brief" request.
    - If hi.Sctl.ca (help -control_arg control) is TRUE, calls help_util_$execute to run the "control_arg" request.
    - If hi.Sctl.title (help -titles control) is TRUE, calls help_util_$execute to run the "titles" request.
    - If hi.Sctl.lep (help -lep control) is TRUE, calls help_util_$execute to run the "list_entry_points" request.
    - If hi.Sctl.all (help -all control) is TRUE, then:
        - ALL_PARAGRAPHS: is a label declared in display_block at the beginning of this do-group.
        - Code in this do-group actually displays all sections of the current block.
        - If the current block is a subroutine introduction, it also displays all sections of each subroutine entry point block.
- If hi.Sctl.all (help -all control) or hi.Sctl.bf (help -brief control) or hi.Sctl.ca (help -control_arg control) is TRUE, then all requested data from this block has been displayed.
    - Returns to the caller of display_block.

- BLOCK_DISPLAY_BEGINS: is a label declared in display_block at the point where info block sections get displayed.
    - Sets hi.ANOTHER_BLOCK_LABEL to ANOTHER_BLOCK_OF_SAME_INFO_SEG label (described below).
    - Sets hi.ALL_PARAGRAPHS_LABEL to the ALL_PARAGRAPHS label (described above).
    - If a header has not been displayed above, displays one now:
        - Calls help_listen_util_$set_iPgh_range to recommend first paragraphs to display. These might be in middle of block if -section or -search were given to select a particular section/paragraph.
        - Calls help_util_$execute to run the header request. This displays a header line, and sets hi.block_progress = BLOCK_PROGRESS_display.
    - Calls help_listen_util_$print_iPgh_range to print the first (or selected) section(s) from the block.
    - Calls ssu_$listen to enter the loop (described in "ssu_ Listener below) which continues displaying info section(s) a page at a time, with "More help?" prompts allowing user to control what's printed.
        - ssu_$listen runs in an infinite loop. help_listen_util_$display_prompt will exit this loop when all block paragraphs have been seen by doing a non-local transfer to hi.NEXT_BLOCK_LABEL.
        - Other response routines can also do a non-local transfer to exit the loop: `next`, `quit`, `entry_point`, `find_entry_point`

ANOTHER_BLOCK_OF_SAME_INFO_SEG: is a label in the display_block routine providing one of these transfer points. This label is assigned to hi.ANOTHER_BLOCK_LABEL. Its code:

- Prints several blank lines to separate the new block's paragraphs from those of the earlier block.
- Sets hi.another_blockS to TRUE, indicating that a short block header should be displayed.
- Then transfers to the BLOCK_DISPLAY_BEGINS label shown above.

A response routine like `entry_point` or `find_entry_point` shifts to a different subroutine entry block as follows:

- Searches for an entry point iBlok, given either its name or a STR argument to search for.
- Sets hi.iBlokP to point to that entry point's iBlok structure.
- Transfers to hi.ANOTHER_BLOCK_LABEL (which has the value ANOTHER_BLOCK_OF_SAME_SEG).

A response routine like `all_paragraphs` or `all_entry_points` displays all sections of the info block or subroutine info segment by transferring to hi.ALL_PARAGRAPHS_LABEL.

help_$term Entry Point Changes

The existing help_$term performs the following steps to clean up the help_ invocation:

- If the Phelp_args pointer is null, then help_$init did not complete any initialization steps; return without taking any steps.
- If unspec(help_args.help_data_ptr) is all zero bits, help_$init acquired the temporary segment holding help_args, but did not complete its initialization; just release the temporary segment.
- Otherwise:
  - If help_args.help_data_ptr is not null, call help_$term recursively with that pointer as the Phelp_args argument, to release the secondary help_ invocation created to display help for help's responses.
  - Then release the temporary segment containing the help_args structure.

The revised help_$term can perform fewer steps. Now that help_$init creates its own cleanup on-unit to manage interruptions of the initialization process, help_$term does not have to deal with the case of a partially-initialized help_args structure (the second bullet above). Nor does it have to deal with any recursive invocation of help_ (the fourth bullet above). The revised help_$term does the following:

- If Phelp_args pointer is null, then help_$init did not fully complete its initialization steps; return without taking any steps.
- Otherwise:
  - Use the Phelp_args pointer to locate the help_info (hi) structure (at the very beginning of the temporary segment containing help_args).
  - If hi.isdP is not null, call info_seg_$terminate with that pointer to terminate the info_seg_ environment.
  - If hi.sciP is not null, call ssu_$destroy_invocation to clean up the ssu_ operating environment created by help_$init. This releases all areas and temporary segments managed by ssu_, including the temporary segment holding the help_args structure.
  - Set Phelp_args pointer to null and return to the caller.

help_ Shared Procedures: error, ioa, newline, response_error

One issue arises in many parts of the help_ code: ensuring there are a specified number of blank lines before and/or after particular help output (info paragraphs, or info heading lines, or help about responses, etc.). Use of a special newline-generating procedure resolves this issue.

Syntax: call newline (desired_NL_count);

This procedure uses the hi.help_numbers.newline_Nblanks_output variable to record how many consecutive NL characters have been output since the last non-blank line was displayed. It emits desired_NL_count – hi.newline_Nblanks_output NL characters whenever it is called.

This algorithm depends on mechanisms for outputting non-blank lines to set hi.newline_Nblanks_output to zero. To accomplish this, all non-error-related output generated by help_ and its support routines is generated by calls to a special ioa_-like procedure.

Syntax: call ioa (ioa_control_string, arg1, …, argN);
call ioa$nnl (ioa_control_string, arg1, …, argN);

This procedure is called only to output non-blank lines. Its first step is to check hi.print_inhibitS: if TRUE, then it returns without displaying anything. Its second step is to set hi.newline_Nblanks_output to zero. Then it calls ioa_ (or ioa_$nnl) with its argument list to generate the output. The hi.print_inhibitS switch is set in the on-unit for the "program_interrupt" condition to suppress current printing; it is cleared just before printing the next user prompt.

Two kinds of error-related output come from help_ and its support routines. Methods for reporting errors must also set hi.newline_Nblanks_output to zero, since lines of an error message represent non-blank output. Both error procedures have a calling sequence like ssu_$print_message.

Syntax: call error (sci_ptr, status_code, ioa_control_string, arg4, …, argN);
call response_error (sci_ptr, status_code, ioa_control_string, arg4, …, argN);

The error procedure is called to report problems with the input arguments to help_$help_. The response_error procedure is called to report problems with the arguments provided with a help_ response (in the response to the "More help?" prompt). Both check that help_args.Sctl.inhibit_errors is FALSE. Both call newline (1) to ensure that a blank line precedes the error message. Both set hi.newline_Nblanks_output to zero, to sure that output following any error message (e.g., the next user prompt) is preceded by a blank line.

The response_error procedure then sets hi.prompt_repeatS to TRUE, and calls ssu_$abort_line to display the error message. This ensures that any user response following the response reporting an error will not be evaluated, and that the prior "More help?" prompt line will be repeated.

The error procedure calls ssu_$print_message to display the error message.

All four of these special procedures are included in _help_shared_data_.incl.pl1, because help_ and each of its support source programs make calls to two or more of these procedures; and because the procedures depend upon data in the help_info structure which is declared in that include file.

ssu_ Listener

The most important ssu_ service is provided by ssu_$listen. This invokes the ssu_ listener to perform the steps implemented by the existing help_rql_ subroutine to display the sections of an info block. In particular, ssu_$listen performs the following flow-of-control steps as an infinite loop.

1. ssu_$listen calls a **pre-request-line hook** (stub procedure) before reading each request. help_ replaces this hook with: **help_listen_util_$display_prompt**. This subroutine does the following:
    a. If all sections of the current info block have been seen by the user:
        i. Do a non-local goto that unwinds out of ssu_$listen and returns back to hi.NEXT_INFO_LABEL, the help_ code to display the next item in the "selected info blocks" list.
    b. Otherwise:
        i. Use display state and current paragraph position (hi.selected) within the info block to decide which info paragraphs to display next.
        ii. Record the selected range of paragraphs in hi.iPgh_print_range.
        iii. Display a "More help?" prompt describing section titles in the chosen print range.
        iv. Set hi.prompt_repeatS = TRUE (where TRUE = "1"b) to force a repeat of the prompt if an empty response (blank response line) is given.
        v. Return to ssu_$listen.

2. ssu_$listen then reads a request line (user response to the "More help?" prompt). It processes the response as follows.
    a. Parse the response line into one or more requests (where a semi-colon separates each request from any that follow in the response line).
    b. Parse each request into request name and arguments.
        i. Lookup request name in the active ssu_ request table(s).
        ii. If request name is not found, call an **unknown_request** procedure to report the unknown request name to the user. help_ replaces this procedure with: **help_requests_$unknown_response**. This pseudo-response procedure calls response_error to diagnose the unknown request name using a more help-like message.
        iii. If the request name is found, construct an argument list from any remaining strings in the request; then call the found request (response procedure) with that argument list.

3. Each invoked ssu_ request performs one of the following actions.
    a. Approve display of the selected range of sections (e.g., the yes response).
        i. Set hi.prompt_repeatS to FALSE ("0"b).
        ii. Return to ssu_$listen.

b. Select another info section to display (e.g., the `section`, `search`, `top` or `unseen_paragraphs_only` responses).
    i. Store that selection in hi.selected.
    ii. Set hi.iPgh_print_range to null.
    iii. Set hi.prompt_repeatS to FALSE.
    iv. Return to ssu_$listen.
c. Select another info block to display (e.g., the `entry_point`, `find_ep` or `find_info`, or `no` or `next` responses).
    i. Store that selection in hi.iBlok.
    ii. Do a non-local goto the hi.ANOTHER_BLOCK_LABEL or hi.NEXT_BLOCK_LABEL which exits out of ssu_$listen.
d. Perform some other action (e.g., `help TOPIC`, `brief`, `control_arg` or `list_entry_points` responses), which displays a different set of data requested by the user.
    i. Leave hi.prompt_repeatS TRUE.
    ii. Return to ssu_$listen.
e. Exit from the help_$help_ subroutine call (e.g., the `quit` response).

4. ssu_$listen then calls a **post-request-line hook**. Replace this hook with: **help_listen_util_$print_iPgh_range**. This procedure does one of the following:
    a. If hi.prompt_repeatS is TRUE, print nothing and return to ssu_$listen.
    b. Otherwise does the following.
        i. Check if hi.iPgh_print_range pointers are null. If so, call help_listen_util_$set_iPgh_range which uses hi.selected pointers to choose a range of paragraphs to print. This changes hi.iPgh_print_range from the info sections listed in the "More help?" response to info sections selected by the most recent response routine.
        ii. Print the range of paragraphs in hi.iPgh_print_range.
        iii. Return to ssu_$listen.

5. ssu_$listen repeats these steps in an infinite loop by jumping to step (1) above.

Before beginning the infinite loop above, ssu_$listen establishes a program_interrupt on-unit. This on-unit calls a **program_interrupt hook**. Replace this hook with an internal procedure of help_.pl1 called **help_program_interrupt_**. This short procedure:

- Sets hi.prompt_repeatS TRUE, to ensure a prompt is displayed.
- Sets hi.print_inhibitS TRUE to ensure other data is not displayed until ready for the prompt.
- Returns to ssu_$listen's program_interrupt on-unit.

The ssu_$listen program_interrupt on-unit then branches to ssu_$listen's READ_REQUEST label, step (1) of the infinite loop above.

Additional information about the help_listen_util_ subroutine is provided in the next section.

help_listen_util_ Subroutine

ssu_$listen provides the flow-of-control loop for displaying paragraphs of an info block, as described in the *ssu Listener* section above. Steps 1 and 4 of that loop (summarized below) call hooks that may be replaced by a subsystem to add functions in this loop. The help_listen_util_ subroutine provides those replacement routines.

1. Before each request line is read, ssu_$listen calls a pre-request-line hook. help_ replaces this hook with the **help_listen_util_$display_prompt** entry point. It decides which info paragraphs would be *most useful to the user*, constructs a "More help?" prompt giving title and length of sections containing these paragraphs, and displays that prompt.
2. ssu_$listen then reads the user response to that prompt, which it processes as an ssu_ request line. The listener breaks this line into one or more requests with arguments; then invokes each request in sequence.
3. For help_, these request procedures are *response routines* which can select different paragraphs from those named in the prompt.
4. After each request line is processed, ssu_$listen calls a post-request-line hook. help_ replaces this hook with the **help_listen_util_$print_iPgh_range** entry point which actually displays either: the recommended paragraphs in the response; or different paragraphs selected by the user's response to the prompt.

Later subsections describe these two hook replacement routines in more detail.

help Use-Cases

The choice of *paragraphs most useful to the user* is based on the following use-case analysis.

- A user invokes help to get some specific information. The control arguments given with help hint about which information is desired.

  help -brief
  help -control_arg

  hint that the user wants specific information about syntax details, and nothing else.

  RECOMMENDATION: help_ should just display that information and stop.

  help -titles

  hints that the user isn't sure what type of information is desired, or where it might be found in the info block.

  RECOMMENDATION: help_ should display section titles, and prompt about information in only the first section of the info block.

help -section STRs
help -search STRs

hint that the user has some idea about what information is desired, and the terminology related to that information.

RECOMMENDATION: help_ should search for section/paragraph matching those STR arguments, and begin with that matching section, starting with any matching paragraph.

- Once display of paragraphs begins, the user expects paragraphs to be displayed in their order of appearance in the info block.

  RECOMMENDATION: help_ should prompt about paragraphs immediately following the last paragraph actually displayed. If those next paragraphs seem uninteresting (based upon the section titles given in the prompt), the user response can select a different paragraph to display.

- When the last paragraph has been displayed, the user may be unsure if every paragraph of the info block has been displayed.

  RECOMMENDATION: If some paragraphs are unseen, the prompt should alert the user that “End of info” has been reached but some paragraphs remain. The prompt should describe the first “unseen” paragraphs. Subsequent prompts should describe only the remaining “unseen” paragraphs.

- When displaying an info block describing a subroutine, additional use-cases come into play.

help SUBROUTINE_

Giving a subroutine name without a particular entry point hints that names or functions of particular entry points may be unknown to the user.

RECOMMENDATION: help_ should begin displaying the subroutine introduction block which usually begins a subroutine info segment. help_ ensures it always includes a list of entry point names. The introduction block is usually short enough to fit on one page. So help_ should automatically begin displaying the first entry point, printing just its Function section then prompting for “More help?”.

RECOMMENDATION: When reaching the end of an entry point block, help_ should continue with the next unseen entry point block, printing just its Function section then prompting for “More help?”.

help SUBROUTINE_ -ep
help SUBROUTINE_$ENTRYNAME

Giving an explicit entry point name hints that only information about that entry point is needed.

RECOMMENDATION: help_ should begin displaying that entry point block. When reaching end of that entry point block, help_ should check for and display any unseen paragraphs in that entry point block, then stop.

help_ State Information

The help_listen_util_$display_prompt entry point uses state information for the current info block being displayed to determine which starting paragraph to recommend in the next prompt, and how many paragraphs to include in the next print range. The following data items are used in making this determination.

iPgh.line_count
: Count of lines in the paragraph described by each iPgh structure.

iPgh.seenS
: Switch set TRUE in each iPgh structure whenever that paragraph is displayed by help_listen_util_$print_iPgh_range.

iBlok.seenS
: Switch set TRUE in each iBlok structure whenever any paragraph is that block is displayed by help_listen_util_$print_iPgh_range.

hi.display_mode
: Integer determining mode for display_block operations. It has one of the values:
  - DISPLAY_MODE_non_subroutine: displaying a non-subroutine info block
  - DISPLAY_MODE_subroutine: displaying a block in a subroutine info seg
  - DISPLAY_MODE_help_request_loop: displaying a block describing one of the help_ responses.

hi.display_limit
: Integer determining what type and how many paragraphs to display. It has one of the values:
  - DISPLAY_LIMIT_none: display all paragraphs that fit on the video terminal page.
  - DISPLAY_LIMIT_section: display at most one or more paragraphs in a single section.
  - DISPLAY_LIMIT_unseen: display only unseen paragraphs.
  - DISPLAY_LIMIT_rest_unseen: display all remaining unseen paragraphs.

help_ maintains a set of pointers that identify the **current paragraph**. Four pointers maintain this state.

hi.iFileP
: pointer to iFile structure describing the info segment containing the current paragraph.

hi.iBlokP
: pointer to iBlok structure describing the info block containing the current paragraph.

hi.selected.iPghP
: pointer to iPgh structure describing the **current paragraph**.

hi.selected.iSectP
: pointer to iSect structure describing the section containing the current paragraph.

The hi.iFileP pointer changes only when display_block jumps to help_'s NEXT_INFO label, to move on to the next item in the "selected info blocks" array.

The hi.iBlokP pointer changes only when switching from one subroutine entry point block to another, or when using the `find_info` or `info` responses to choose a different block in a non-subroutine info segment.

The two hi.selected pointers are the main state identifiers. They work together as a pair. Their meaning changes slightly as ssu_$listen progresses through the four steps of the ssu_ listener loop.

When entering step 1:

```
 If  hi.selected.iPghP  ^= null,   it points to the "last paragraph printed".

 But if switching to a new section or paragraph...
  If hi.selected.iPghP   = null  &
     hi.selected.iSectP  = null:   “last paragraph printed" appears to be a
                                   paragraph before 1st section of block.
  If hi.selected.iPghP   = null  &
     hi.selected.iSectP ^= null:   “last paragraph printed" appears to be
                                   the paragraph before that section.
```

```
Step 1:  Uses "last paragraph printed", hi.display_limit, and count of unseen
         paragraphs to determine what paragraphs to display next; then
         tells user what they are...  via a "More help?" prompt.  One of its
         final operations changes hi.selected to point to the upcoming
         “current paragraph” (the next paragraph recommended for display).

    [ssu_$listen calls help_listen_util_$display_prompt to perform step 1.]
```

When entering step 2 and for all subsequent steps:

```
 hi.iPgh_print_range.startP   points to first iPgh recommended for display.
                    .endP     points to last  iPgh recommended for display.


 hi.selected.iPghP   = hi.iPgh_print_range.startP
                              "current paragraph" is first paragraph
                               recommended for display.
           .iSectP   = hi.iPghP->iPgh.relatives.sectP
                               pointer to iSect describing section containing
                               "current paragraph".
 hi.prompt_repeatS   = TRUE:   Step 4 should print nothing; then the next
                               step 1 selects/displays a possibly-new
                               "More help?" prompt.
```

```
Step 2:  ssu_$listen reads and processes the user's response to that
         "More help?" prompt.
              - It break's response line into one or more ssu_ requests.
              - Invokes each request (usually a help_ response routine).
              - If user types an empty response line, the hi.prompt_repeatS=T
                setting causes prior prompt to be redisplayed.
```

```
Step 3:  Each help_responses_ routine can adjust hi.selected and
         hi.iPgh_print_range, or may make no adjustments: these values
         select which info paragraphs are actually displayed.

         A response routine may also change hi.display_limit to control
         how many or which paragraphs are displayed within the print range.

         A response routine (e.g., list_entry_points or ?) may itself display
         data other than info paragraphs.

         To display info paragraphs in step 4, a response routine must set:

   hi.prompt_repeatS = FALSE:  either the incoming hi.iPgh_print_range is
                               displayed, or an adjusted hi.selected and/or
                               hi.iPgh_print_range displayed.
```

```
When entering step 4:

 If hi.iPgh_print_range  ^= null:  hi.iPgh_print_range.startP and .endP
                                   delimit the exact set of paragraphs to
                                   display.
                          = null:  if .startP is null, then revised
                                   hi.selected values specify a new
                                   "current paragraph".

                In the = null case, help_listen_util_$set_iPgh_range is
                called to map that "current paragraph" into a new pair of
                hi.iPgh_print_range.startP and .endP pointers.

                If hi.display_limit = DISPLAY_LIMIT_section, that revised
                print range is limited to paragraphs of the current
                paragraph's section, from the current paragraph onward.

Step 4:  If hi.prompt_repeatS = FALSE, then display paragraphs of the
         hi.iPgh_print_range.
          - If hi.display_limit >= DISPLAY_LIMIT_unseen, then
            only unseen paragraphs in print range are displayed.

     [ssu_$listen calls help_listen_util_$print_iPgh_range to perform step 4.
      After performing its work, it returns back to ssu_$listen...
      which iterates again through LOOP.... starting at step 1.]
```

help_listen_util_$display_prompt

The help_listen_util_$display_prompt entry point decides which paragraphs of the current info block would be most useful to display to the user now; then builds a multi-line prompt giving section title(s) and line count(s) included in that print range; and finally displays the prompt ending with the question: More help?

The *most useful paragraph to display* is usually the paragraph immediately following the last displayed paragraph. However, when the last paragraph of the info block is displayed, help_ enters an "unseen paragraphs only" state by setting hi.display_limit = DISPLAY_LIMIT_unseen. In this state, only unseen paragraphs are displayed starting with the first unseen paragraph in the info block.

Therefore, the first operation by $display_prompt is to check:
Is hi.display_limit >= DISPLAY_LIMIT_unseen?

This tests also checks for the DISPLAY_LIMIT_rest_unseen value. If in this state, the operation counts how many unseen paragraphs exist in the entire block, and sets a first_unseen_pghP pointer to the first unseen paragraph's iPgh structure.

- If unseen paragraphs are found, the first unseen paragraph is assigned as the starting point in the print range being built (iPghP = first_unseen_pghP). This iPghP designates the "current paragraph" being examined for possible display.
- If no unseen paragraphs are found, exit this block. This operation jumps to the next item in the "selected info blocks" array; or for a subroutine info segment entered via the introduction block (user did know specify a particular entry point), it jumps to the next unseen entry point block in this subroutine info seg (if any are unseen).

If not in an unseen paragraphs state, then the code sets an iPghP pointer to the iPgh structure following the "last displayed paragraph" value. This iPghP designates the "current paragraph" being examined for possible display.

- If iPghP is null, that means the last paragraph in the block was most recently displayed. The code counts unseen paragraphs in the block, setting first_unseen_pghP to point to the iPgh structure for the first unseen block.
    - If unseen paragraphs are found, iPghP points to the first unseen paragraph in the block. hi.display_limit is set to DISPLAY_LIMIT_unseen.
    - If no unseen paragraphs are found, exit the block as described above.

The second operation by $display_prompt is to reserve a certain number of lines at the bottom of each terminal page for "More help?" prompts. This is known as the **prompt frame**. Line space is apportioned to the prompt frame based upon the length of the user's terminal screen, as returned by the get_page_length_ function.

- The minimum prompt reservation is three lines, used if total page length is 15 lines or less.
- The maximum prompt reservation is seven lines, used if total page length is more than 35 lines.
- The first reserved prompt line is always a blank line separating the prompt from earlier info paragraphs.
- 50 characters are reserved in the last prompt line to hold the user's response to the "More help?" prompt.
- The remaining lines on the page are available for display of info paragraphs. These lines are known as the **info frame**.

The third operation by $display_prompt is to determine how many consecutive paragraphs will fit in the info frame. Several factors affect this determination. The hi.iPgh_print_range.startP and .endP values point to the first and last consecutive paragraphs recommended for display.

- The paragraph is the smallest unit of information considered in this decision. All lines of a paragraph must be displayed at the same time.
- The goal is to display full sections of information on the user's terminal page without triggering the output I/O module's "more" prompt: "More? (RETURN for more; DEL to discard output.)".
  - However, some sections may be longer than the info frame. These longer sections must be split into groups of paragraphs that will fit on one page, and then displayed one group at a time.
- Once the last group of paragraphs in a section fits in the info frame, additional sections can be selected for display if the following constraints are met.
  - All paragraphs of an added section must fit in the info frame.
  - The section's title and line count must fit in the prompt frame.

The fourth operation by $display_prompt is to set hi.selected.iPghP and .iSectP to point to the first iPgh structure in the print range, and its containing section's iSect structure.

The fifth operation by $display_prompt is to actually display the "More help?" prompt describing the sections next recommended for display. The user can reply "yes" to have those lines displayed; or may give other responses to select other information.

Two internal procedures assist in these operations.

some_section_paragraphs_will_fit
: determines what part of the current section under consideration will fit in the info frame. It returns TRUE if any parts fit, along with a part-of-section indicator. Or it returns FALSE if no parts of the current section will fit.

prompt_add_will_fit
: determines whether a section title and line count can be added to a "More help?" prompt being built. If the new section info fits, it is added to the array of prompt lines.

help_listen_util_$set_iPgh_range

The help_listen_util_$set_iPgh_range entry point decides which paragraphs of the current info block would be most useful to display to the user now, based upon the current help_ state information. It is called in two situations.

- Before the ssu_ listener loop is entered, $set_iPgh_range is called to select the first section(s) to display from each info block.
- In step 4 of the ssu_ listener loop when help_listen_util_$print_iPgh_range is called with hi.iPgh_print_range = null, it is called to select specific paragraphs to display given the display state. This can occur when a response routine in step 3 sets hi.selected values to choose a different starting point for the next paragraphs to display.

In either situation, $set_iPgh_range performs the first four operations of the help_listen_util_$display_prompt entry point to set the hi.iPgh_print_range.startP and .endP.

help_listen_util_$print_iPgh_range

The help_listen_util_$print_iPgh_range entry point prints the info paragraphs actually selected for display (either those recommended by the $display_prompt or $set_iPgh_range entry points; or those chosen by a help response routine).

if hi.prompt_repeatS = TRUE, this routine immediately returns to the ssu_ listener.

If hi.iPgh_print_range.startP is null, code calls the $set_iPgh_range entry point to set the print range, based upon the current help_ state information. Otherwise, it uses the print range set in earlier steps of the ssu_ listener loop.

If hi.display_limit ^= DISPLAY_LIMIT_rest_unseen, the code displays all paragraphs in the print range.

If hi.display_limit = DISPLAY_LIMIT_rest_unseen, the code finds the first unseen paragraph in the given print range, and displays that paragraph and all subsequent unseen paragraphs within the print range. In such cases, the print range usually ends with the last paragraph of the info block.

For either hi.display_limit case, all paragraphs displayed have their iPgh.seenS and iBlok.seenS bits set to TRUE.

Finally, the code sets hi.selected.iPghP to point to the final iPgh structure for the last displayed paragraph. hi.selected.iSectP points to its containing section's iSect structure. Both hi.selected pointers are set to null if the print range extended beyond the final paragraph of the info block. hi.iPgh_print_range is set to null. If hi.display_limit = DISPLAY_LIMIT_section, then selection of only blocks in the current section is complete; hi.display_limit is set to DISPLAY_LIMIT_none.

help_responses_ Subroutine

ssu_$listen provides the flow-of-control loop for displaying paragraphs of an info block, as described in the *ssu Listener* section above. In Step 3 of that loop, the listener invokes a *response routine* to the "More help?" prompt. The prompt describes help_'s recommendation for the next info section(s) to display. Each response routine implements one of five possible user reactions to that recommendation.

A. Approves display of the selected range of sections (e.g., the `yes` response).
B. Selects another info section to display (e.g., the `section`, `search`, `top` or `unseen_paragraphs_only` responses).
C. Selects another info block to display (e.g., the `entry_point`, `find_ep` or `find_info`, or `no` or `next` responses).
D. Performs some other action which displays a different set of data requested by the user, then repeats the "More help?" prompt (e.g., `help TOPIC`, `brief`, `control_arg` or `list_entry_points` responses).
E. Exits from the help_$help_ subroutine call (e.g., the `quit` response).

This *help_responses_ Subroutine* section describes the response routines that display information from the current info block, or select some other info block to display.

The subsequent section entitled *help_requests_ Subroutine* describes other response routines that implement more generic ssu_-related requests.

Within each of these sections, the response routines are described in alphabetic order by primary response name. Each routine must:

- Process any input arguments and control arguments. The ssu_ support routines arg_setup and args_remain (from ssu_request_dcls_.incl.pl1) and ssu_$arg_ptr are called to process input parameters.
- Implement the response action.

The following responses are supported when displaying a non-subroutine info block.

```
List of responses for most infos:
 yes, y
 rest {-section | -unseen} {-top},
    r {-scn | -uns} {-t}
 skip {-section | -seen | -rest},
    s {-scn | -seen | -rest}
 next, no, n
 quit, q

 brief, bf
 control_arg {STRs} {-case_sensitive},
    ca {STRs} {-cs}

 titles {-top | -unseen},
  title {-t | -uns}
 section {STRs} {-case_sensitive} {-top},
    scn  {STRs} {-cs} {-t}
 search  {STRs} {-case_sensitive} {-top},
    srh  {STRs} {-cs} {-t},
    sh   {STRs} {-cs} {-t}

 top {-section}, t {-scn}
 header, he

 unseen_paragraphs_only, unseen, uns
 every_paragraph_visible, every_pgh, every

 find_info {STRs} {-case_sensitive} {-top},
    find  {STRs} {-cs} {-t}
 info {INFO_NAME} {-list|-ls}
 all_paragraphs, all

 help {RESPONSE_NAME} {-brief | -control_arg STR},
    h {RESPONSE_NAME} {-bf | -ca STR}
 list_responses, list_requests, lr
 ?
 .
 .. MULTICS_COMMAND_LINE
```

The following responses are supported when displaying a subroutine info block.

```
List of responses for subroutine infos:
 yes, y
 rest {-section | -unseen} {-top},
    r {-scn | -uns} {-t}
 skip {-section | -seen | -rest}
    s {-scn | -seen | -rest},
 next, no, n
 quit, q

 brief, bf
 control_arg {STRs} {-case_sensitive},
    ca {STRs} {-cs}

 titles {-top | -unseen},
  title {-t | -uns}
 section {STRs} {-case_sensitive} {-top},
    scn  {STRs} {-cs} {-t}
 search  {STRs} {-case_sensitive} {-top},
    srh  {STRs} {-cs} {-t},
    sh   {STRs} {-cs} {-t}

 top {-section}, t {-scn}
 header, he

 unseen_paragraphs_only, unseen, uns
 every_paragraph_visible, every_pgh, every

 list_entry_points {-unseen},
    list_ep {-uns},
    lep {-uns}
 entry_point {-match | -exact} {EP_STR} {-unseen},
    ep {-match | -exact} {EP_STR} {-uns}
 find_entry_point {STRs} {-case_sensitive} {-top},
    find_ep {STRs} {-case_sensitive} {-top},
    find {STRs} {-cs} {-t}
 all_entry_points, aep, all

 help {RESPONSE_NAME} {-brief | -control_arg STR},
    h {RESPONSE_NAME} {-bf | -ca STR}
 list_responses, list_requests, lr
 ?
 .
 .. MULTICS_COMMAND_LINE
```

Response: all_paragraphs, all, all_entry_points, aep

Reaction B: jump to the ALL_PARAGRAPHS label in display_block, which: displays all paragraphs of the current info block; or for a subroutine info seg, displays paragraphs describing all entry points of the subroutine.

Response Names:	all_paragraphs, all	(DISPLAY_MODE_non_subroutine)
	all_entry_points, aep, all	(DISPLAY_MODE_subroutine)
Syntax as a response:	all_paragraphs all	(DISPLAY_MODE_non_subroutine)
	all_entry_points aep all	(DISPLAY_MODE_subroutine)

Calls arg_setup and args_remain to verify no arguments or control arguments were given with this request.

Sets hi.Sctl = FALSE, and hi.Sctl.all = TRUE, which selects the display_block opening code for the help -all control argument. For a subroutine info segment, sets hi.iBlokP to point to the iBlok for the first block of the info segment (normally, the subroutine introduction block). This causes the -all code to display descriptions for that introduction and every entry point.

Jumps to hi.ALL_PARAGRAPHS_LABEL, which exits the ssu_ listener loop and unwinds to the ALL_PARAGRAPHS label of display_block. Once the info block or subroutine info segment has been displayed, the hi.Sctl.all switch causes display_block to return to its caller: the next item in the help_$help_ “selected info blocks” array.

Response: brief, bf

Reaction D: displays Syntax-related section(s) of current block; and item names found in Arguments…, Control arguments…, and List of… sections.

Response Names:	brief, bf
Syntax as a help control:	-brief -bf
Syntax as a response:	brief bf

Calls arg_setup and args_remain to verify no arguments or control arguments were given with this request.

Checks if a header has been displayed for the current info block. If not, calls the `header` request.

Calls info_seg_util_$is_Syntax_type for each section of the current block. If the section is one of the Syntax… type of sections, calls help_util_$print_section[3] to display all lines in that section.

Calls info_seg_util_$is_List_type for each section of the current block. If the section is one of the Arguments…, Control arguments…, or List of … type of section, calls help_util_$print_iSect_List to display a columnized list of item names described in that section.

Returns to the ssu_ listener without changing hi.prompt_repeatS, so the last "More help?" prompt is repeated.

Response: control_arg, ca

Reaction D: displays details about items found in Arguments…, Control arguments…, and List of… sections whose item name matches one of the STRs parameters to this response.

Response Names:	control_arg, ca
Syntax as a help control:	-control_arg STRs {-case_sensitive} -ca STRs {-cs}
Syntax as a response:	control_arg {STRs} {-case_sensitive} ca {STRs} {-cs}

Checks hi.CACHE_caP ^= null to access STRs from most recent control_arg response, or from -control_arg values given in help_args.cas array (help -control_arg operands). Does *delayed-allocation* of the ca_cache_store structure if hi.CACHE_caP = null, and copies any help_args.cas elements into that cache.

Calls arg_setup and args_remain to look for arguments or control arguments given with this request. Calls Args_ca() routine to copy those response parameters into the ca_cache_store structure.

Complains to user if no new parameters were given, and no saved parameters exist from an earlier control_arg response for this block.

Checks if a header has been displayed for the current info block. If not, calls the `header` request.

Calls info_seg_util_$is_List_type for each section of the current block. If the section is one of the Arguments…, Control arguments…, or List of … type of section, calls help_util_$print_iSect_List_match to display lines describing item names matched by STRs arguments.

Complains to user if no matching items found.

Returns to the ssu_ listener without changing hi.prompt_repeatS, so the last "More help?" prompt is repeated.

[3] The help_util_$print_section routine does not set the iPgh.seenS switch. Displaying the section via brief response does not mark that section as having been seen by the user.

Response: entry_point, entrypoint, ep

Reaction C: searches for an :Entry: block in current info segment having an entry point name matching EP_STR parameter; or the next unseen entry point if -unseen is given. If more than one entry point matches EP_STR: displays a list of matching entry point names; then asks the user to give the ep response with an exact EP_STR argument from that list.

Response Names: entry_point, entrypoint, ep (DISPLAY_MODE_subroutine)

Syntax as a response: entry_point {-match | -exact} {EP_STR} {-unseen}
entrypoint {-match | -exact} {EP_STR} {-uns}
ep {-match | -exact} {EP_STR} {-uns}

Calls arg_setup and args_remain to look for arguments or control arguments given with this request. Calls Args_ep() routine to copy those response parameters into the ep_store structure.

Checks the first block of the subroutine info segment to obtain the reference name portion of the subroutine's entry point names. If user included the reference name in any of the EP_STR values, removes that reference name from the EP_STR value.

If neither EP_STR nor -unseen was given, looks for an info block with entry point name which is the same as the subroutine's reference name (e.g., entry point name of help_ for the help_$help_ entry point).

Checks for a subroutine info segment having only one block (e.g., match_star_name_.info) describing an entry point whose entry point name is the same as the subroutine reference name. If its entry point name matches the given (or default) EP_STR, leaves that block as selected. Otherwise, calls response_error to report no entry point matches the given EP_STR.

For normal subroutine info segments, checks all blocks following the subroutine introduction to see if its matches the EP_STR, or is unseen if -unseen was given. Calls response_error if no matching block was found. If one matching block was found, switches to that block. If -unseen was given with no EP_STR, switches to the first unseen entry point block. If several matching blocks were found, lists their names and asks the user to reissue the ep response with a more exact EP_STR argument.

If a block switch is needed: sets hi.iBlokP to the selected entry point block; sets hi.selected and hi.iPgh_print_range to null to begin at the top of that block; sets hi.display_limit to DISPLAY_LIMIT_section to display only the Function section of that entry point when the block is first entered, and sets hi.selected.iSectP to point to the first iSect structure in that block; sets hi.prompt_repeatS to FALSE; and jumps to the hi.ANOTHER_BLOCK_LABEL.

Response: every_paragraph_visible, every_pgh, every

Reaction D: Reverts from displaying only the unseen paragraphs of the current block to displaying all paragraphs.

Response Names: every_paragraph_visible, every_pgh, every

Syntax as a response: every_paragraph_visible
every_pgh
every

Calls arg_setup and args_remain to verify no arguments or control arguments were given with this request.

Sets hi.display_limit to DISPLAY_LIMIT_none. Sets hi.iPgh_print_range to null, so a new set of recommended sections to display is built. Ensures hi.prompt_repeatS is TRUE, so nothing is printed before the revised prompt is displayed.

Returns to the ssu_ listener.

Response: find_ep, find_info, find

Reaction C: Searches forward from the current block to find the next block with heading line or paragraph containing one of the STR strings. The first matching block becomes the current block being displayed. When invoked to implement help -search or -section control arguments for a subroutine info segment, searches for a subroutine entry point description containing a STR string given with those controls. The first matching entry point is selected for display.

Response Names: find_info, find (DISPLAY_MODE_non_subroutine)
find_ep, find (DISPLAY_MODE_subroutine)

Syntax as a help control: -search STRs {-case_sensitive}
-srh STRs {-cs}
-section STRs {-case_sensitive}
-scn STRs {-cs} (DISPLAY_MODE_subroutine)

Syntax as a response: find_info {STRs} {-case_sensitive} {-top}
find {STRs} {-cs} {-t} (DISPLAY_MODE_non_subroutine)
find_ep {STRs} {-case_sensitive} {-top}
find {STRs} {-cs} {-t} (DISPLAY_MODE_subroutine)

Checks hi.CACHE_findP ^= null to access STRs from most recent find response, or from -search and/or -section values given in help_args.srh and/or help_args.scn arrays. Does *delayed-allocation* of the find_cache_store structure if hi.CACHE_findP = null, and copies any help_args.srh and help_args.scn elements into that cache.

Calls arg_setup and args_remain to look for arguments or control arguments given with this request. Calls Args_find() routine to copy those response parameters into the find_cache_store structure.

If -top was given, begins searching in the first block of the info segment. Otherwise, begins search with the block following the current info block.

Searches each block's heading line and paragraphs looking for a match with any of the STR strings. If -top was given, skips the current info block.

If a match is found: sets hi.section_search_matchedS to TRUE; sets hi.iBlokP to point to the matching block's iBlok structure; sets hi.selected and hi.iPgh_print_range to null (positioning to "top" of that matching block); and jumps to the hi.ANOTHER_BLOCK_LABEL.

If no matching block was found: reports an error; sets hi.section_search_matchedS to FALSE; leaves hi.prompt_repeatS set to TRUE; and returns to the ssu_ listener.

Response: header, he

Reaction D: Displays a header line describing: current info segment; current info block with count of total lines in block; count of lines recommended for next section to display (if any).

Response Names:	header, he	
Syntax as a help control:	-header, -he	(displays only a heading line)
	-brief_header, -bfhe	(displays brief heading lines with info paragraphs)
Syntax as a response:	header he	

Calls arg_setup and args_remain to verify no arguments or control arguments were given with this request.

Despite its simple syntax, the header response routine provides a complex service. It displays different info block heading line formats for each of the many cases in which a header can be displayed. Some headers are a single line of output; others involve several lines of data.

The simplest header format is displayed for an info segment which matches a help_args.path(J) starname, but whose name duplicates that of an info segment found earlier in the info_segment search paths. Such identically-named info segments are listed below a title:

```
Other versions of the info above were found. See also:
  PATHNAME
  PATHNAME
```

Heading lines displayed for non-subroutine info blocks differ from those for a subroutine introduction block, or a subroutine entry point block.

Heading line format depends on the progress made in displaying the current info block.

- If help_info.Sctl.he_only is TRUE (set by help -header control), only a heading line is displayed for the info block; no paragraphs are displayed.
  PATHNAME [Info: INFO_NAME] HEADING_LINE (LINES_IN_INFO_BLOCK)

- If just starting to display the info block (hi.block_progress = BLOCK_PROGRESS_needs_header), then the heading line is omitted from the header, but is displayed on a separate line.
  PATHNAME [Info: INFO_NAME] (LINES_FOLLOW; LINES_IN_INFO_BLOCK)

  DATE HEADING_LINE

- If just entering a different block of the current info segment (e.g., `find_info` or `info` response selected a different block), a shorter heading is displayed:
  Info: INFO_NAME (LINES_IN_INFO_BLOCK)

- If user entered the `header` response, the displayed header always includes the full pathname:
  PATHNAME [Info: INFO_NAME] (LINES_IN_INFO_BLOCK)

For subroutine info blocks, the LINES_IN_INFO_BLOCK strings also include data about the number of subroutine entry points described in the info segment.

- If just entering a different block of the current info segment (e.g., (e.g., `ep` or `find_ep` response selected a different block), a shorter heading is displayed:
  Entry: ENTRY_POINT_NAME (LINES_IN_INFO_BLOCK; COUNT_OF_OTHER_ENTRY_POINTS)

Response: info

Reaction C: If INFO_NAME is given, searches for an :Info: or :[Info]: block in current info segment having a block name exactly matching the INFO_NAME parameter. Selects that matching block for display.

Reaction D: If -list is given, lists all blocks in the current info segment.

Response Names:	info	(DISPLAY_MODE_non_subroutine)
Syntax as a response:	info {INFO_NAME} { -list | -ls }	

Calls arg_setup and args_remain to look for arguments given with this request. Calls Args_info() routine to copy the parameters into the info_store structure.

If -list or -ls was given, lists divider names for all blocks in the current info segment.

If an INFO_NAME was given, checks all blocks of the info segment for a block divider name matching INFO_NAME. Calls response_error if no matching block was found. If one matching block was found, switches to that block.

If a block switch is needed: sets hi.iBlokP to the selected info block; sets hi.selected and hi.iPgh_print_range to null to begin at the top of that block; sets hi.prompt_repeatS to FALSE; and jumps to the hi.ANOTHER_BLOCK_LABEL.

Response: list_entry_points, list_ep, lep

Reaction D: Displays a list of subroutine entry point heading lines.

Response Names: list_entry_points, list_ep, lep (DISPLAY_MODE_subroutine)

Syntax as a help control: -list_entry_points, -lep

Syntax as a response: list_entry_points {-unseen}
list_ep {-uns}
lep {-uns}

Calls arg_setup and args_remain to look for arguments given with this request. Calls the Args_lep() routine to copy the parameters into the lep_store structure.

Then it gets the subroutine reference name from the first word of the heading line for the current subroutine block.

The response routine prepares to build an entry point heading LIST structure at the end of help_'s temporary segment. If -unseen was given, it adds heading for any unseen entry point blocks into the LIST; and columnizes the lines of the LIST.

If -unseen was not given, it references a full entry point heading LIST structure allocated by an earlier call to help_requests_$lep_setup.

The routine then calls the header response routine if hi.block_progress = BLOCK_PROGESS_needs_header and then calls help_util_$print_LIST to display the selected LIST structure.

Finally, it returns to the ssu_ listener with hi.prompt_repeatS still TRUE.

help_responses_$lep_setup

This utility entry point delay-allocates a LIST structure whose items are heading lines from a subroutine info segment's entry point blocks. hi.storage.LIST_entrypointsP records location of this LIST structure. It then ensures that the subroutine introduction block ends with an "Entry points in…" section (adding that section if it is missing. Finally, it calls either info_seg_parse_$replace_STRING_in_section or info_seg_parse_$append_STRING_to_block to populate lines of the "Entry points in…" section with lines from the columnized LIST structure.

Response: next, no, n

Reaction C: Returns from the display_block internal procedure so the next item in help_'s "selected info blocks" array may be displayed.

Response Names: next, no, n

Syntax as a response: next
no
n

Calls arg_setup and args_remain to verify no arguments or control arguments were given with this request.

Jumps to hi.NEXT_INFO_LABEL, which does non-local transfer out of the ssu_listener and display_block, to help_'s NEXT_INFO label.

Response: rest, r

Reaction B: revises the recommended print range.

Response Names:	rest, r
Syntax as a response:	rest {-section | -unseen } {-top} r { -scn | -uns } {-t}

Calls arg_setup and args_remain to look for arguments given with this request. Calls the Args_rest() routine to copy the parameters into the rest_store structure.

The rest_store.rest_mode is a trinary switch which has one of the following values:

- REST_MODE_none: rest was given without -section or -unseen controls.
- REST_MODE_section: rest was given with -section control argument.
- REST_MODE_unseen: rest was given with -unseen control argument.

The recommended print range is revised based on the rest_mode value, presence of the -top control argument, and the current hi.display_limit setting.

If rest_mode is REST_MODE_none, then:

- If hi.display_limit = DISPLAY_LIMIT_unseen (help is displaying only unseen paragraphs of current block), then assumes rest_mode is REST_MODE_unseen (see paragraph below).
- If -top was given, sets hi.iPgh_print_range.startP to point to first paragraph of the current info block. Otherwise, starts display with the first paragraph recommended for printing.
- Sets hi.iPgh_print_range.endP to point to last paragraph of the current info block.
- Sets hi.prompt_repeatS to FALSE so paragraphs are printed.
- Sets hi.clear_prompt_regionS to TRUE, so the most recent prompt and the users response are removed from the terminal scroll-back buffer.
- Returns to the ssu_ listener with the modified print range.

If rest_mode is REST_MODE_section, then:

- If -top was given, sets hi.iPgh_print_range.startP to point to the first paragraph of the current section. Otherwise, starts display with the first paragraph recommended for printing.
- Sets hi.iPgh_print_range.endP to point to the last paragraph of the current section.
- Sets hi.prompt_repeatS to FALSE so paragraphs are printed.
- If all paragraphs are being displayed (hi.display_limit < DISPLAY_LIMIT_unseen), then sets hi.clear_prompt_regionS to TRUE, so the most recent prompt and the users response are removed from the terminal scroll-back buffer.
- Returns to the ssu_ listener.

If rest_mode is REST_MODE_unseen then:

- If -top was given, sets hi.iPgh_print_range.startP to point to the first paragraph of the current info block. Otherwise, starts display with the first paragraph recommended for printing.
- Sets hi.iPgh_print_range.endP to point to last paragraph of the current info block.
- Sets hi.display_limit to DISPLAY_LIMIT_rest_unseen, which causes only unseen paragraphs in the print range to be displayed; seen paragraphs in that range are skipped.
- Sets hi.prompt_repeatS to FALSE so paragraphs are printed.
- Returns to the ssu_ listener.

Response: section, scn

Reaction B: Sets the current paragraph to be the first paragraph of the next section of the info block whose title matches one of the STR strings. If -top is given, the section hunt starts with the first section of the info block.

If the help controls are used, the matching section becomes the first section of the info block to be displayed. But if no matching section is found, the entire info segment is skipped (ignored in the search for the help_args.path(J) name within the info segment search paths). Thus -section participates in selecting info blocks to display. This matches the actions of the existing help_ subroutine.

```
Response Names:          section, scn

Syntax as a help control:        -section  STRs   {-case_sensitive}
                                     -scn  STRs   {-cs}

Syntax as a response:             section {STRs} {-case_sensitive} {-top}
                                      scn {STRs} {-cs} {-t}
```

Checks hi.CACHE_scnP ^= null to access STRs from most recent section response, or from -section values given in help_args.scn arrays. Does *delayed-allocation* of the scn_cache_store structure if hi.CACHE_scnP = null, and copies any help_args.scn elements into that cache.

Calls arg_setup and args_remain to look for arguments or control arguments given with this request. Calls Args_scn_srh() routine to copy those response parameters into the scn_cache_store structure.

If -top was given, begins searching in the first section of the info block. Otherwise, begins search with the section following the current info section.

Searches each section title looking for a match with any of the STR strings.

If a match is found:

- sets hi.section_search_matchedS to TRUE;
- sets hi.selected.iSectP to point to the matching section's iSect structure;
- sets hi.selected.iPghP to null, positioning to "top" of that matching section;
- sets hi.iPgh_print_range to null so the recommended print range will be recalculated;
- sets hi.prompt_repeatS to FALSE, so the section will be displayed;
- sets hi.display_limit = DISPLAY_LIMIT_section so only the matching section is displayed; and
- returns to the ssu_ listener.

If no matching block was found:

- if invoked as the section request, reports an error;
- sets hi.section_search_matchedS to FALSE;
- leaves hi.prompt_repeatS set to TRUE; and
- returns to the ssu_ listener.

Response: search, srh

Reaction B: Sets the current paragraph to be the next paragraph of the info block containing a match for one of the STR strings. If -top is given, the searching starts with the first paragraph of the info block.

If the help controls are used, the matching paragraph becomes the first paragraph of the info block to be displayed. But if no matching paragraph is found, the entire info segment is skipped (ignored in the search for the help_args.path(J) name within the info segment search paths). Thus -search participates in selecting info blocks to display. This matches the actions of the existing help_ subroutine.

Response Names: search, srh

Syntax as a help control: -search STRs {-case_sensitive}
-srh STRs {-cs}

Syntax as a response: search {STRs} {-case_sensitive} {-top}
srh {STRs} {-cs} {-t}

Checks hi.CACHE_srhP ^= null to access STRs from most recent search response, or from -search values given in help_args.srh arrays. Does *delayed-allocation* of the srh_cache_store structure if hi.CACHE_srhP = null, and copies any help_args.srh elements into that cache.

Calls arg_setup and args_remain to look for arguments or control arguments given with this request. Calls Args_scn_srh() routine to copy those response parameters into the srh_cache_store structure.

If -top was given, begins searching in the first paragraph of the info block. Otherwise, begins search with the paragraph following the current paragraph.

Searches each line of the paragraph looking for a match with any of the STR strings.

If a match is found:

- sets hi.section_search_matchedS to TRUE;
- sets hi.selected.iPghP to point to the iPgh structure of the paragraph preceding the matching paragraph and set hi.selected.iSectP to point to that paragraph's containing section; or if the matching paragraph is first in block, sets hi.selected = null to select the "top" of the block.
- sets hi.iPgh_print_range to null so the recommended print range will be recalculated;
- sets hi.prompt_repeatS to FALSE, so the section will be displayed;
- returns to the ssu_ listener.

If no matching block was found:

- if invoked as the search request, reports an error;
- sets hi.section_search_matchedS to FALSE;
- leaves hi.prompt_repeatS set to TRUE; and
- returns to the ssu_ listener.

Response: skip, s

Reaction B: revises the recommended print range.

Response Names: skip, s

Syntax as a response: skip { -section | -seen | -rest }
s { -scn | -seen | -rest }

Calls arg_setup and args_remain to look for arguments given with this request. Calls the Args_skip() routine to copy the parameters into the skip_args structure.

The skip_args.skip_mode is a quad switch which has one of the following values:

- SKIP_MODE_none: skips paragraphs in the recommended print range, and prompts about sections/paragraphs that follow that range.
- SKIP_MODE_section: skips paragraphs in the first section of the recommended print range, and prompt about sections beyond that first section.
- SKIP_MODE_seen: skips to and prints the next unseen paragraph; sets hi.display_limit to DISPLAY_LIMIT_section so only that unseen paragraph and the remainder of its section are displayed.
- SKIP_MODE_rest: skips to the bottom of the info block; prompts if some paragraphs were not seen by the user.

The recommended print range is revised based on the skip_mode value and the current hi.display_limit setting.

If skip_mode is SKIP_MODE_none, then:

- If hi.display_limit = DISPLAY_LIMIT_unseen (help is displaying only unseen paragraphs of current block), then assume skip_mode is SKIP_MODE_unseen (see paragraph below).
- Selects the final paragraph of the recommended print range as the "last printed paragraph": hi.selected.iPghP, iPghP = hi.iPgh_print_range.endP; and sets hi.selected.iSectP to point to that paragraph's section structure: hi.selected.iSectP = iPgh.relatives.sectP.
- Sets hi.prompt_repeatS to TRUE so no paragraphs are printed and the recommended print range is reevaluated.
- Returns to the ssu_ listener with the modified print range.

If skip_mode is SKIP_MODE_section, then:

- Sets the "last paragraph printed" to the final paragraph in the first section of the recommended print range:
  hi.selected.iSectP, iSectP = hi.iPgh_print_range.startP->iPgh.relatives.sectP;
  hi.selected.iPghP = iSect.relatives.pghs.lastP;
- Sets hi.prompt_repeatS to TRUE so no paragraphs are printed and the recommended print range is reevaluated.
- Returns to the ssu_ listener.

If skip_mode is SKIP_MODE_seen then:

- Walks through paragraphs from start of recommended print range to last paragraph in the current block, looking for an unseen paragraph. If found:
  - Sets the current paragraph to that first unseen paragraph.
  - Sets the current section to its section.
  - Sets hi.iPgh_print_range.startP to point to that first unseen paragraph.
  - Sets hi.iPgh_print_range.endP to point to last paragraph of its section.
  - Sets hi.prompt_repeatS to FALSE so this revised print range is displayed.
  - Returns to the ssu_ listener.
- If not found, assumes skip_mode is SKIP_MODE_rest.

If skip_mode is SKIP_MODE_rest then:

- Sets the "last paragraph printed" (hi.selected.iPghP) to point to the last paragraph of the block.
- Sets its section (hi.selected.iSectP) to the last paragraph's section structure (iPgh.relatives.sectP).
- Sets hi.prompt_repeatS to TRUE so the recommended print range is reevaluated when after the "end of the info block".
- Returns to the ssu_ listener.

Response: titles, title

Reaction D: Displays section titles remaining in the current info block; or all section titles if -top is given; or all section titles containing unseen paragraphs if -unseen is given.

Response Names: titles, title

Syntax as a help control: -titles, -title

Syntax as a response: titles { -top | -unseen}
title { -t | -uns }

Calls arg_setup and args_remain to look for arguments given with this request. Calls the Args_title() routine to copy the parameters into the title_args structure.

Applies -top or -unseen to determine starting section for the list of titles.

Gets scratchpad storage for a temporary LIST structure in help_'s temporary segment. Copies section titles from starting point to end of info block into the LIST structure, with an appropriate title reflecting the starting point.

Calls help_util_$format_one_use_LIST to columnize entries in the list. Calls help_util_$print_LIST to display the list.

Abandons storage for the LIST structure at the end of the temp segment.

Returns to the ssu_ listener.

Response: top, t

Reaction B: Sets the current paragraph to be either: the first paragraph of the current section, if -section was given; or the first paragraph of the info block. If -section was given, displays the entire section; otherwise builds and displays a prompt about sections at the top of the block, and displays that prompt.

Response Names: top, t

Syntax as a response: top {-section}
t {-scn}

Calls arg_setup and args_remain to look for arguments given with this request. Calls the Args_top() routine to copy the parameters into the title_args structure.

Sets hi.display_limit to DISPLAY_LIMIT_none if the display was restricted only to the current info section.

If -section was given: sets hi.iPgh_print_range to point to all paragraphs of the current section; sets hi.prompt_repeatS to FALSE, causing that section to be displayed.

Without -section: sets hi.selected to null, positioning the “last printed paragraph” before the first section of the info block; sets hi.iPgh_print_range to null; prints the block heading line to re-identify the block to the user; sets hi.prompt_repeatS to TRUE, which builds and displays a prompt describing first sections of the block.

Returns to the ssu_ listener.

Response: unseen_paragraphs_only, unseen, uns

Reaction B: Sets the current paragraph to the first paragraph of the info block. Sets state to display only unseen paragraphs. Builds and displays a revised prompt describing the first unseen paragraph in the block.

Response Names: unseen_paragraphs_only, unseen, uns

Syntax as a response:
```
unseen_paragraphs_only
                unseen
                   uns
```

Calls arg_setup and args_remain to verify no arguments or control arguments were given with this request.

Sets hi.selected to null to select the first paragraph at top of the info block. Sets hi.iPgh_print_range to null to clear any current print range. Sets hi.display_limit to DISPLAY_LIMIT_unseen to display only unseen paragraphs.

Sets hi.prompt_repeatS to TRUE to build and display a revised prompt describing the first unseen paragraph.

Returns to the ssu_ listener.

Response: yes, y

Reaction B: Sets the current paragraph to the first paragraph of the info block. Sets state to display only unseen paragraphs. Builds and displays a revised prompt describing the first unseen paragraph in the block.

Response Names: yes, y

Syntax as a response:
```
yes
  y
```

Calls arg_setup and args_remain to verify no arguments or control arguments were given with this request.

Sets hi.prompt_repeatS to TRUE to build and display a revised prompt describing the first unseen paragraph.

If hi.display_limit is displaying every paragraph (not in unseen_paragraphs_only state), sets hi.clear_prompt_regionS to TRUE to remove prompt and any user reply from the terminal scroll-back buffer.

Returns to the ssu_ listener.

help_requests_ Subroutine

This section describes other response routines that implement more generic ssu_-related requests. Because these responses display output or can report errors, they must be implemented using help_'s ioa, error, newline, and response_error routines.

Response: quit, q

This response routine exits the help_ subsystem.

Response Names: quit, q

Syntax as a response: quit
q

Calls arg_setup and args_remain to verify no arguments or control arguments were given with this request.

The routine exits the help_ subsystem by calling ssu_$abort_subsystem. This ssu_ subroutine causes the ssu_ listener to return to its caller (help_'s display_block) with the status code ssu_et_$subsystem_aborted.

When receiving that status code, display_block does a non-local goto help_'s SUBSYSTEM_ABORTED label which cleans up the help_$help_ invocation and returns to its caller.

Response: . (self-identify)

This response routine display's the help_ subsystem version number. For the ssu_ implementation of help, the value displayed is: help version: 2.00

Response Names: **.**

Syntax as a response: **.**

Calls arg_setup and args_remain to verify no arguments or control arguments were given with this request.

Returns to the ssu_ listener.

Response: ? (summarize_responses)

This response routine displays abbreviated syntax for all responses accepted at the "More help?" prompt. The list of response names displayed depends upon the hi.display_mode value, since a different ssu_ request table is setup for DISPLAY_MODE_subroutine versus DISPLAY_MODE_non_subroutine modes.

Response Names: **?**

Syntax as a response: **?**

Calls arg_setup and args_remain to verify no arguments or control arguments were given with this request.

Checks for a previously created LIST structure describing responses for the current display mode. hi.LIST_non_subroutine_responsesP points to the LIST structure for non-subroutine responses. hi.LIST_subroutine_responsesP points to the LIST structure for subroutine-specific responses. If the given pointer is null, then a new LIST structure is delay-allocated in the help_ temporary segment and initialized. help_util_$format_LIST is called to columnize that list.

Finally, calls help_util_$print_LIST to display the list.

Returns to the ssu_ listener.

Response: help, h

This response routine provides meta-information (second-level online help information) about the help_ subsystem and its response routines.

Response Names: help, h

Syntax as a response: help {RESPONSE_NAME} { -brief | -control_arg STR}
h {RESPONSE_NAME} { -bf | -ca STR }

While it may seem like help_$help_ is being invoked a second time to display info about help responses, the actual implementation of the help response routine does all the display tasks itself.

If no RESPONSE_NAME is given with the help response, the response routine calls the summarize_responses routine to display a columnized list of the acceptable responses. This is followed with canned advice paragraphs telling how to display more details about a particular help response.

If a RESPONSE_NAME is given, the help response routine does the following:

- Looks for a help.info segment: in the same directory containing the help_ (bound_info_rtns_) subroutine; or else in >doc>info directory.
- Calls info_seg_$examine_iFile and info_seg_$parse_iFile to parse that info segment into blocks, sections, and paragraphs.
- Searches for an internal info block (one using the :[Info]: block divider token) with the given RESPONSE_NAME as its block divider name.
- Does one of the following:
    - If -brief is given: summarizes Information from that block by calling the `brief` response routine.
    - If -control_arg is given: prints information about control argument(s) matching STR by calling the `control_arg` response routine.
    - Otherwise, displays the full block by calling help_util_$print_section for each section in that block.

The help response never issues its own "More help?" prompt. It always displays all of the requested info block, then returns to the ssu_ listener.

The example below shows entering a "help brief" response (asking for data about the help_ subsystem's brief response) at a "More help?" prompt. Data about the brief response is surrounded by demarcation lines to separate that second-level online help from the paragraphs of info block being displayed by the display_block call. Such demarcation lines surround all data displayed by a "help RESPONSE_NAME" response.

```
Control arguments (selecting info segments) (15 more)
  &  Control arguments (info displayed) (38)
  &  Control arguments (paragraph grouping) (6)
  &  Notes on control arguments (4).  More help?  help brief

  /* --  help response:  2020-10-27  brief, bf  ------------------

Syntax:  brief, bf

Function:  prints a summary of a command, active function or
subroutine info seg including its Syntax section and item names in
"Arguments", "Control arguments", and "List of ..." sections.  Then
repeats the previous "More help?" question.

     ------------------------------------------------------------ */

Control arguments (selecting info segments) (15 more)
  &  Control arguments (info displayed) (38)
  &  Control arguments (paragraph grouping) (6)
  &  Notes on control arguments (4).  More help?
```

help_requests_$unknown_response

In step 3 of the ssu_ listener loop, the listener breaks the user's response line into separate responses (divided from one another by semi-colon characters), and parses each response into the name of a response (a request_name in ssu_ terminology), and arguments given to that response (if any are present).

The listener then looks in the current ssu_ request table(s) for an entry describing the given request_name. If no matching entry is found, the user has typed an invalid response name. The listener calls a replaceable **unknown_request** subroutine to report this invalid response name to the user.

help_ replaces the default unknown_request subroutine with its own help_requests_$unknown_response subroutine. That subroutine reports the error to the user using help_-specific terminology (referring to an "Unknown response" instead of an "Unknown request"), and using the response_error routine to issue the actual error message.

Because response_error calls ssu_$abort_line to print the error message, the help_requests_$unknown_response never actually returns to the ssu_ listener. ssu_$abort_line unwinds ssu_ listener's call to $unknown_response, and aborts processing of the remainder of the user's response line.

help_util_ Subroutine

The help_util_ subroutine provides a variety of services to other parts of help_ and its response routines. These services include the following.

help_util_$count_file_lines
: Calculates the *printable line* count for an info segment. This is called to obtain the printable lines in an info segment describing subroutines. [See the definition of *printable line count* in the *Glossary*.]

help_util_$execute
: Calls ssu_$execute_string to invoke a response routine with optional arguments and control arguments. Handles return codes from ssu_$execute_string indicating that a "program_interrupt" condition was signaled during execution of the response routine; or that a request to abort the subsystem occurred while executing the response routine.

help_util_$print_iSect_List
: Displays names for items appearing in an "Arguments…", "Control arguments…" or "List of…" section of an info segment in columnized format. Called to produce output for the brief response.

help_util_$print_iSect_List_match
: Displays names and definition for items appearing in an "Arguments…", "Control arguments…" or "List of…" section of an info segment that match selector STR values given in a control_arg response.

help_util_$print_section
: Displays printable lines in one section of an info block, preceded and followed by (at most) one blank line.

Two services support organizing short data items into one or more columns. Items to be columnized are stored in a LIST structure declared in _help_shared_data_.incl.pl1:

```
dcl 1 LIST aligned based, /* structure used by help_util_ to format list of things  */
    2 header,             /* to be output in columns.                               */
      3 (N,               /*   number of list elements.                             */
         Npghs,           /*   number of filled paragraphs of formatted out.        */
         Nrows,           /*   number of rows in formatted output.                  */
         Ncols,           /*   number of columns in formatted output.               */
         ML (6)           /*   length of longest element in each column.            */
         ) fixed bin,
     2 title char (84),   /*   title of output list (includes COLON ending title).  */
                          /*    NOTE: This element is pointed to by iLine.P & .L,   */
                          /*          and by the Line structure.  It cannot be a    */
                          /*          varying string.                               */
     2 group (0 refer (LIST.N)),
       3 arg char (88) varying,
                          /*   the data item.                                       */
       3 Snot_found fixed bin,
                          /*   = 1 if no match found for the argument.              */
     2 print_array (0 refer (LIST.Npghs), 0 refer (LIST.Nrows)),
       3 line_out char (HELP_LINE_SIZE_MAX);
                          /* Array of lines to be output to the screen.             */
```

help_util_$format_LIST

Formats items in the LIST.arg(*) array into a multi-paragraph, multi-column table with items organized as up to six descending varying-width columns of items. The resulting column are stored within lines of the LIST.line_out(*, *) array.

Each LIST structure is *delay*-allocated in help_'s temporary segment. Once the list is fully populated with data, the set_space_used subroutine is called to complete the delay-allocate operation.

While the code supports multi-paragraph formats, all callers specify formatting the data into a single paragraph (even if it exceeds the recommended maximum paragraph length of 15 lines).

help_util_$format_one_use_LIST

Works like help_util_$format_LIST but does not complete the delay-allocate operation. The LIST is stored in scratch-pad storage, and is abandoned when the caller of the $format_one_use_LIST returns to its caller.

help_util_$print_LIST

Prints the LIST.line_out(*, *) lines. Each list is preceded by a blank line.

info_seg_ Subroutine Changes

Three new internal entry points are being added to the info_seg_ subroutine[4] for use by help_. Since all three entry points are internal to (not retained by) bound_info_rtns_, they are not documented in the info_seg_.info file. They are described in code comments preceding each entry point. This commented usage is included below to describe these internal entry points in this bulletin.

info_seg_$init_for_help_

The first new entry point provides a quick alternative for the info_seg_$initialize entry point. Each call to the help_$help_ entry point passes the info_seg_data structure (initialized by help_$init) to this new entry point. info_seg_data.sciP points to help_'s ssu invocation pointer (hi.sciP), so the same ssu_ invocation may be shared by help_ and the info_seg_ subsystem. In addition, info_seg_data.std_areaP points to help_'s standard area pointer (hi.areaP), so this area may be shared for allocation/free operations by both subsystems.

To these pointers, the info_seg_$init_for_help_ entry point adds info_seg_data.areaP which points to a translator_temp_ expandable, no-freeing area. This area is owned and managed by the info_seg_ subsystem. It holds storage for *info segment structure hierarchy* items describing contents of each info segment. These hierarchy items are allocated and owned by the info_seg_ subsystem, but are shared for reading by help_ and its help response routines and utilities.

```
Syntax:
   declare info_seg_$init_for_help entry (ptr, fixed bin(35);
   call info_seg_$init_for_help (info_seg_dataP, code);

Arguments:
info_seg_dataP
   points to the info_seg_data structure (see info_seg_dcls_.incl.pl1)
   which help_ provides as input.
     A) help_ sets info_seg_data.version to indicate which structure
        version it is presenting.
     B) help_ sets info_seg_data.ptrs = null() to prepare for a later
        call to info_seg_$terminate
     C) help_ sets info_seg_data.ptrs.isdP to an ssu_ sci_ptr value
        created by the caller.
     D) help_ optionally sets info_seg_data.ptrs.std_areaP to an area
        with "standard characteristics" setup by the caller's ssu_
        invocation.
   See the "Notes on cleanup" section for more information.

code
   is a status code.  It is non-zero when a fatal error occurs.
   Cases include:
    - info_seg_data.version has a value not supported by info_seg_.
```

[4] Retained entry points of the info_seg_ subroutine are described in *MTB-1004: verify_info Command & info_seg_*.

```
Notes on cleanup:
When initialization completes, info_seg_data.ptrs point to data items
used in subsequent calls to info_seg_ subroutines.  The caller must
create a cleanup on-unit that calls a routine to release storage for
these data items:

   - Calls info_seg_$terminate to terminate info segments and release
     storage holding structures describing those info segments.

In addition, the caller must call the above routine before the command
returns.
```

info_seg_$examine_iFile

The second new entry point performs parts of the service provided by info_seg_$append_iFiles, and part of the service provided by info_seg_$parse_iFile. Since help_ has already handled star convention expansion of an incoming pathname and identified specific file(s) to be displayed, it passes in location and file unique identifier (UID) information for one info segment.

The new entry point stores those details in an iFile structure, initiates the file, and partially parses the file into lines and block component structures which are threaded from that iFile structure. It then threads the iFile onto the info_seg_data structure, and returns a pointer to that iFile to help_.

```
Syntax:
   declare info_seg_$examine_iFile entry (ptr, char(*), char(*),
       bit(36) aligned, fixed bin(35)) returns (ptr);
   iFileP = info_seg_$examine_iFile (info_seg_dataP, dir, ent,
                                     uid, code);

Arguments:
info_seg_dataP
   points to the info_seg_data structure (see
   info_seg_dcls_.incl.pl1). (Input)
dir
   absolute pathname of directory containing info segment. (Input)
ent
   entry name by which info segment was selected from that
   directory. (Input)
uid
   unique identifier for that info segment. (Input)
iFileP
   points to a new iFile structure for that info segment, or to an
   existing iFile structure if the info segment was already added via
   an earlier call to this function. (Output)
code
   standard status code. (Output)  It reports any error encountered
   while initiating the file.

Notes on iFile structures:
Each iFile structure created by info_seg_ is added to the
info_seg_data.files threaded list.  Use the following code to set
iFileP to point to structures on this list:

     do iFileP = info_seg_data.files.firstP
          repeat iFile.nextP  while (iFileP ^= null() );
             ... <code to operate on each iFile> ...
          end;
```

help_ adds data about the info segment structure of each file, location of the info block selected by the input data, and other information to its “selected info blocks” array. This data is sufficient for help_ to determine which of several matching info segments having a given name should actually be displayed.

If help_$help_ is given several pathnames, or a starname matching many info seg names, info_seg_$examine_iFile is called for each matching file. To avoid having many info segments initiated in the process at one time, help_ limits the number of initiated files to 10. Before attempting to get data for an 11th info segment, help_ calls info_seg_$reinitialize which terminates the earlier info segments, and removes their iFile structures from the info_seg_data threads. It then empties the translator_temp_ storage area of all prior storage, to reduce amount of process directory storage needed for help_.

When help_ is actually ready to display an info segment, it calls info_seg_$examine_iFile a second time to access data for the file. If only a few info segments were examined earlier, this call finds info seg contents already parsed by that earlier call; it returns the existing iFile structure. If more than 10 files were examined earlier, then info_seg_ must re-initiate the info segment and repeat parsing its contents into lines and blocks.

To complete parsing of a given iFile, help_ then calls the regular info_seg_$parse_iFile entry point. This adds iSect and iPgh structures to each iBlok found in the info segment.

info_seg_$reinitialize

The third new entry point terminates all info segments initiated by earlier info_seg_ calls, removes the iFile structures describing these info segments from the info_seg_data structure thread, and empties the translator_temp_ area in which other structures of the *info segment structure hierarchy* are allocated by info_seg_$examine_iFile and info_seg_$parse_iFile.

```
Syntax:
   declare info_seg_$reinitialize entry (ptr);
   call info_seg_$reinitialize (info_seg_dataP);
```

help_$help_ calls this entry point before calling info_seg_$examine_iFile if more than 10 iFile structures are found on the info_seg_data.files thread. It also calls the entry point at its SUBSYSTEM_ABORT label before returning to its caller.

info_seg_parse_ Subroutine Change

Two new internal entry points are being added to the info_seg_parse_ routine, which is an internal subroutine of the info_seg_ subsystem. They are called by help_responses_$lep_setup to insert an actual list of subroutine entry points into the "Entry points in…" section of a subroutine introduction block. This insertion of text is done so special software is not needed to display this fabricated info section.

info_seg_parse_$append_STRING_to_block

If a subroutine info segment includes an introduction block that does not have a template "Entry points in…" section, then help_responses_$lep_setup constructs the full text of such section as a STRING, and passes that STRING to the $append_STRING_to_block entry point, along with a pointer to an iBlok structure for the introduction block.

The info_seg_parse_$append_STRING_to_block routine: copies this STRING into info_seg_ storage; constructs a paragraph from lines of the string; appends that paragraph to a new iSect structure; then appends that iSect structure to the iBlok structure describing the introduction block.

info_seg_parse_$replace_STRING_in_section

If a subroutine info segment includes an introduction block that already has a template "Entry points in…" section, then help_responses_$lep_setup constructs the full text of such section as a STRING, and passes that STRING to info_seg_parse_$replace_STRING_in_section entry point, along with a pointer to an iBlok structure for the introduction block and the iSect structure for the template "Entry points in…" section.

The info_seg_parse_$replace_STRING_in_section routine: copies this STRING into info_seg_ storage; constructs a paragraph from lines of the string; removes existing paragraphs ("List is generated by the help command") from the template "Entry points in…" section; appends the new paragraph(s) to that section; then updates the iBlok.line_count for the introduction block.

Test Strategies

The info_seg_ subsystem for parsing info segments has been extensively tested while preparing the verify_info (vi) command. So there is little need to test how help_ parses an info segment into blocks, sections, paragraphs, and lines.

Testing will therefore be focused on the command interface (changes to the help command) and the subsystem interface (changes to help_ entry points and help_args_.incl.pl1); testing of the help_$help_ internal code; testing of help_'s replacement procedures in the ssu_ listener; and the help_ response routines.

Accuracy of documentation for the help command, help_ subroutine, and help_ response routines will also be verified against the code.

These testing strategies are described more fully in the next few subsections.

Command Interface Testing

The help_ subsystem is most often invoked using the help command. Changes in this command were kept to a minimum, as summarized in the earlier *help Command Changes* section of this MTB. Functional testing is needed only for the new or changed control arguments implemented by these changes.

Subsystem Interface Testing

The help_ subsystem is used by other Multics subsystems to display online help information. Most of these subsystems invoke help as an ssu_ request (ssu_requests_$help). ssu_requests_.alm maps that name onto the help$ssu_help_request entry point. Parameters for that interface did not change. So only minimal checks are needed to ensure these calling subsystems still have a working help request.

Testing should verify: that the help_args structure is setup properly based upon arguments passed to the help request shared by all these subsystems; and that the info segment location provided by each subsystem's call-back subroutine is properly specified in help_args.path(J) elements.

Most subsystems using the ssu_requests_$help request reference it by configuring the standard ssu_request_tables_$standard_requests:

```
pcref ssu_request_tables_
References to ssu_request_tables_$standard_requests:  (bound_ssu_ in sss)
    analyze_multics.pl1, deckfile_manager.pl1, dial_out_.pl1, forum.pl1,
    kermit.pl1, linus.pl1, login_server_overseer_.pl1,
    rdm_forward_subsystem_.pl1, rdm_set_request_tables_.pl1,
    read_tape_and_query.pl1, restructure_mrds_db.pl1, sc_create_sci_.pl1,
    sdm_set_request_tables_.pl1
```

Two subsystems reference ssu_requests_$help in their own request tables, rather than by configuring the ssu_request_tables_$standard_requests table:

```
print [library_pathname -library source *req*.alm] -match ssu_requests_$help

                    bound_dm_.3.s.archive::dmsd_ssu_request_tables_.alm

                    ssu_requests_$help,
                    ssu_requests_$help,


                    bound_system_control_.s.archive::sc_request_table_.alm

          request   help,ssu_requests_$help,
```

A few subsystems call the help_ subsystem entry points directly:

```
pcref help_$init
References to help_$init:  (bound_info_rtns_ in STANDARD)
    help.pl1, help_rql_.pl1, mbuild_help_.s::mbuild_help_.pl1,
    probe_info_requests_.pl1, tedhelp_.pl1, tutorial.pl1,
    xforum_help_.pl1, xmail_display_help_.pl1
```

These subsystems must be tested separately to ensure their inputs to help_ are properly handled.

help_ Subroutine Testing

The revised code employs many algorithms copied from the current version of help_. Therefore, results from current and revised code may be compared. Any differences represent either bugs in one of the two implementations, or intentional changes to better serve user needs. Testing should determine the reason for any differences.

The majority of the new code is driven by the highly-stable ssu_ listener code, which requires testing only where its operation is tailored. help_ replaces listener stub procedures with entry points in help_listen_util_.pl1. For details on these changes, see the earlier sections entitled: *ssu_ Listener* and *help_listen_util_ Subroutine*.

The replacement procedures do not control operation of the ssu_ listener; they only display the “More help?” prompt and the info paragraphs described by each prompt (or different data chosen by response routines). So only the displayed data needs to be verified for correctness.

Most testing will focus: on the display_block internal procedure of help_.pl1; on the help_listen_util_ replacement routines; and on the use of help_ state variables that control which paragraphs are displayed.

help_ Response Testing

Compartmentalization of help_ response routines makes it easier to understand and test their options.

Stable methods for processing parameters shared by all response routines require only simple testing of individual parameters accepted by each response.

Actions taken by each response routine must be compared to their intended operations, as stated in the response documentation (help_responses.gi.info, and the per-response info blocks in help.info).

A difficult task is verifying correct name(s) and syntax of each possible response in all the places in which they are documented:

- help_request_tables_.alm (ssu_ request macros), whose information is displayed by the list_responses, list_requests, lr response;
- ? (summarize_responses) output;
- comment at the beginning of list_responses_.pl1;
- help_responses_.pl1 and help_requests_.pl1 code implementing each response and its parameters:
    - comment at beginning of code for each response entry point;
    - actual parameter processing code.
- help.info per-response info blocks, plus the help_responses.gi.info block;
- MTB-1005 documentation for each response.

Installation Plan

The software items described in MTB-1005 are part of a larger product change to replace the existing validate_info_seg (vis) command and help_ subroutine with:

- verify_info (vi): a new command for checking info segments.
- info_seg_verify_: a new subroutine to supervise work of parsing and checking content of info segments.
- info_seg_: a new set of subroutines for parsing info segment contents into components.
- help_ Subsystem: a replacement for the existing help_ and help_rql_ subroutines.

Components of this change are described below.

MTBs

MTB-1004: verify_info Command & info_seg_

MTB-1005: help_ Subroutine (this MTB)

MTB-????: Historical Review of help Facility on Multics

MTB-????: Changes to mbuild Subsystem for Verifying Info Segments

MCRs

MCR10080, Repair validate_info_seg handling of subsystem request infos

MCR10082, Add get_page_length_ subroutine

MCR10083, translator_temp_$empty_all_segments

MCR100??, Copy existing help and validate_info_seg Commands to >obs>bound_old_help_

MCR100??, Replacements for help, help_, validate_info_seg in bound_info_rtns_

MCR100??, Fixes for Info Segments Required to Install verify_info

MCR100??, Add a verify_info Request to mbuild Subsystem and Other mbuild Bug Repairs

MCR10084, Revise probe.info; Enhance probe's help Request

Bugs (Multics Tickets) Not Resolved by these MTBs/MCRs

Ticket 208: validate_info_seg does not handle XXX.compin.info segments correctly

Appendix A: Glossary

Definitions for technical and tactical terms used throughout this MTB are repeated in this glossary for ease of reference.

delayed-allocation

Storage for a based structure is allocated in help_'s temporary segment (assigned a permanent storage location) by a three-step process:

A. **assigning hi.next_free_spaceP** to the pointer locating the structure being allocated;
B. setting any adjustable extents in elements of that structure until it has reached full-size; then
C. **calling set_space_used** with the current size of that structure, which adjusts hi.next_free_spaceP to point to the doubleword just beyond the end of the structure:

```
call set_space_used ( currentsize(based_structure) );
```

The hi.next_free_spaceP points to the first free doubleword aligned storage at the end of the temp segment.

In step B, the based structure can be populated with data, and may grow or shrink in size by setting adjustable extents for character string length or array dimensions.

Permanent allocation only occurs when step C is performed. If the subroutine that performs steps A and B omits step C, the based structure is abandoned when that subroutine returns. Its contents will be overwritten by the next delayed-allocation attempt. The subroutine must ensure that all pointers to the abandoned storage are also released or set to a null value.

info segment structure hierarchy

The ordered set of components found in an online info segment, as defined in *MTB-1004: verify_info Command & info_seg_*. These include the following structures referenced in the MTB-1005.

iFile: defines contents of an entire info segment as a character string with attributes.

iBlok: defines contents of an info block within an info segment as a character string with attributes. Blocks are separated from one another by a block divider line which begins with a divider token (e.g., :Info: or :Entry:, etc.) and includes block divider name(s) that identify the block.

iSect: defines contents of a section within an info segment. Most sections begin with a section title. Many sections contain two or more paragraphs of information text.

iPgh: defines contents of an info paragraph as a character string with attributes. The ideal paragraph contains 15 lines or less of adjacent text, each line holding up to 71 characters. Paragraphs are separated from one another by two blank lines. These blank lines lie outside of any paragraph.

iLine: defines contents of one line of text as a character string. Lines are separated from one another by a newline character. This newline is not part of the character string described by the iLine structure.

printable line count

The count of lines that will be printed for items in the *info segment structure hierarchy*.

block line count: includes the heading line, 1 blank line before each section, plus the section line count for all sections in the block.

section line count: includes the paragraph line count for each paragraph of the section, plus 1 blank line between paragraphs. [Note: only 1 of the two blank lines separating paragraphs is displayed when a section is printed.]

paragraph line count: includes the count of lines within the paragraph. [Note: The 2 blank lines separating one paragraph from another are not included in the line count of any paragraph.]

selected info blocks

An array of items to be displayed by help_$help_. Incoming help_args.path(J) elements may use the star convention to identify several info segments in a directory.

Elements which include no specific directory are searched for using the info_segments (or other caller-specified) search paths. An info segment matching the given entry name may be found in several of the search directories. The segment found in the topmost search directory is the primary info segment to be displayed. Other elements with the same name in lower search directories are called *cross reference segments*. Their Dinfo_seg.Scross_ref switch is TRUE.