Subject: MCR10162, bit_string, virtual_ptr and cv_ptr_, and dump_segment

Author: Gary Dixon

Date: 2026-05-10

Multics Ticket:

https://multics-trac.swenson.org/ticket/373	(bit_string)
https://multics-trac.swenson.org/ticket/375	(dump_segment)
https://multics-trac.swenson.org/ticket/389	(vptr & cv_ptr_)
https://multics-trac.swenson.org/ticket/325	(ssu_)
https://multics-trac.swenson.org/ticket/376	(ssu_)
https://multics-trac.swenson.org/ticket/377	(ssu_)

Introduction

This MCR describes a set of enhancements needed to support exec_com programs which display system data structures, meters, and other support tools. MCR10152 began these improvements by changing ssu_ to enable support for exec_com scripts that invoke analyze_multics requests to display data.

Similar enhancements are needed to access and display per-process and system-wide data segments using both existing and new Multics command / active functions. The enhancements for this MCR are summarized below.

- Ticket_389: recommends adding a virtual_ptr (vptr) command/active function to more easily initiate and terminate data segments being displayed.
    - The vptr command/AF will be implemented as a new entry point in the cv_ptr_ subroutine since these two programs share so much functionality.
    - Reduction statements will be used in the enhanced cv_ptr_ to support a wider variety of VIRTUAL_PTR argument formats using coding methods that are easier-to-implement and support. cv_ptr_.pl1 is already quite complex code. Using reductions will off-load much of the complexity of manipulating the VIRTUAL_PTR string onto software designed to divide input strings into tokens and then parse those tokens systematically.
        - cv_ptr_.pl1 will be renamed cv_ptr_.rd to reflect use of reduction statements. The reductions translator generates PL/1 code to recognize valid VIRTUAL_PTR forms, then assign actions for locating the segment identifier and determining proper offsets within that segment.
    - cv_ptr_.info and virtual_pointers.gi.info will be updated to describe the new VIRTUAL_PTR argument formats.
    - virtual_ptr.info (vptr.info) will be added to describe the new command/AF entry point.

- Ticket_375: suggests that dump_segment (ds) command/AF should be updated to aid in displaying data using commands or active strings in an exec_com. The dump_segment.pl1 implementation currently calls an internal, much older version of the cv_ptr_ subroutine to locate segments to be dumped.
    - Remove this internal cv_ptr_ implementation from dump_segment.pl1.
    - Replace and enhance its functionality by calling a new cv_ptr_$details entry point. This new entry point works like cv_ptr_$cv_ptr_ but also returns details about the segment located by a VIRTUAL_PTR argument to the caller. The dump_segment program needs such details to know:
        - the ring in which the located segment resides -- what method should dump_seg use to read the located data;
        - user's effective access to the located segment – can the user process read data in this segment;
        - size of the located data segment -- how much data can be displayed;
        - full pathname of the segment (which may reside as a component within an archive segment) – dump_segment may need to display pathname of the segment within the file system or entry_name of a wired hardcore segment.

      Other callers of cv_ptr_ may need similar details about the target segment. The new virtual_ptr command could have been implemented separately from cv_ptr_ by using the new cv_ptr_$details interface.
    - Add all the VIRTUAL_PTR formats currently accepted by dump_segment to those supported by cv_ptr_; and document those formats in virtual_pointers.gi.info.
- Ticket_373: requests creation of a bit_string (bit) command / active-function for use in exec_com programs. It will simplify converting bit-oriented data:
    - to a different display radix (binary, quaternary, octal, or hexadecimal); or
    - from a bit_string to an equivalent numeric character-string format – removing double-quote characters surrounding bits of the bit string, along with trailing radix indicator.

- Ticket_325: asks that errors detected in the ssu_ sci_ptr argument be reported more clearly so the user can understand exactly why that pointer has become unusable.
    - Code in the ssu_check_sci procedure (stored in the private _ssu_check_sci.incl.pl1 segment) is called by most parts of ssu_ to validate the incoming sci_ptr provided by the caller.
    - All includers of this include segment will be recompiled to include the new version of ssu_check_sci.
- Ticket_376: asks that two useful ssu_ subroutines be documented in ssu_.info. ssu_$locate_request and ssu_$invoke_request were not included in the MPM Subroutines documentation for ssu_, nor in ssu_.info. Documentation for these two routines will be added to ssu_.info.
- Ticket_377: suggests adding a new routine for dividing character strings expressed using a punctuated language. Valid lexemes found in an input string will be located by a token descriptor for each lexeme. These tokens will be accepted by reductions-based translators as inputs to be translated.

  The lexemes suggested for this new routine are ideal for identifying the various parts of a VIRTUAL_PTR. A VIRTUAL_PTR has the general format:

  SEGMENT_ID $ WordOffset (BitOffset) [RingNo]
  SEGMENT_ID | WordOffset (BitOffset) [RingNo]

    - Add a new ssu_tokens_.pl1 module to bound_ssu_. The new subroutine will include the following entry points: ssu_$get_tokens; ssu_$display_tokens; ssu_$token_id; ssu_$find_token_with_id; and ssu_$get_token_area.
    - Add a new include segment: ssu_token.incl.pl1 to declare data structures for a token descriptor and optional statement descriptor. These descriptors have almost the same content as descriptors provided by the lex_string_ subroutine; and they are compatible inputs to the SYNTAX_ANALYSIS subroutine generated for reductions-based translators.

Lexemes Supported by ssu_tokens_.pl1

An input string is scanned for lexemes (important characters within the input_string) using the following rules.

Scan characters of the input string from left-to-right. Whitespace characters are generally treated as token separators which are not contained within a lexeme. A lexeme may then be defined as follows.

lexeme

is a sequence of characters within the input string which:

- begins with the first non-whitespace character found by the left-to-right scan.
- ends with the first whitespace character, break character, double-quote or list-starting character found during the scan.

whitespace

are characters which separate lexemes from one another. These characters are often not included within any lexeme. Whitespace characters include:

horizontal_tab (HT) space (SP) newline (NL) newpage (NP)

break characters

are characters which break or stop the scan to find the end of the current lexeme. The current lexeme ends with the character preceding each of these break characters. The break character itself is then treated as a separate, 1-character lexeme. The break characters include:

comma (,) colon (:) equal-sign (=) semi-colon (;)

statement

a sequence of lexemes ended by a statement_delimiter lexeme. A statement descriptor (stmt structure) points to the first and last tokens of a statement, and a stmtStr character string overlays the part of input_string contained within the statement.

statement_delimiter

a special lexeme that ends a statement. A semi-colon is often used as the statement_delimiter. But the caller may choose any string of 1 to 4 characters selected from the character-set:

ampersand (&)	dollar-sign ($)	semi-colon (;)
asterisk (*)	exclamation-point (!)	single-quote (')
at-sign (@)	forward-slash (/)	tilde (~)
backward-quote (`)	period (.)	vertical-bar (|)
backward-slash (\)	pound-sign (#)	
circumflex (^)	question-mark (?)	

The statement_delimiter lexeme is treated as a break string that ends any preceding lexeme. Therefore, the delimiter should not include characters expected to appear within other lexemes.

If the statement_delimiter argument is a zero-length (empty string), then statements within the input_string are not identified and no stmt structures are produced by ssu_$get_tokens.

Certain characters used to start a lexeme begin an extended lexeme that may include whitespace or other characters usually treated specially.

quoted string

is a sequence of characters which start and end with a double-quote (") character. All characters between and including the starting and ending double-quote are treated as the quoted string. Two kinds of quoted strings are supported: quoted character string, quoted bit string.

quoted character string lexeme

is a string of characters starting and ending with a double-quote (") character. Any character may appear between the starting and ending double-quote. A pair of consecutive double-quote characters ("") represents one double-quote appearing within the quoted character string.

Example: "A single ""quoted-character"" string."

Each quoted character string token has a token.childP pointing to a token descriptor for its contents: child_tokenStr which is an unquoted string with the surrounding double-quote characters removed; and with any escape double-quote pairs reduced to a single double-quote char.

quoted bit string lexeme

is a string of characters starting with a double-quote (") character and ending with a double-quote character immediately followed by one or two bit-string indicator characters:

INDICATOR	CHARACTERS BETWEEN DOUBLE-QUOTES ARE	
"b or "b1	binary digits	0 1
"b2	quaternary digits	0 1 2 3
"b3	octal digits	0 1 2 3 4 5 6 7
"b4	hexadecimal digits	0 1 2 3 4 5 6 7 8 9 A B C D E F a b c d e f

Examples:	"011"b1	(bit string of length 3)
	"753"b3 = "111101011"b	(bit string of length 9)
	"1A3"b4 = "000110100011"b	(bit string of length 12)

list lexeme

is a sequence of characters within the input string which:

- begins with a particular starting character, and
- ends with a particular ending character.

Unlike a quoted character string lexeme, contents of a list lexeme (excluding its start and end characters) may be scanned again if list_rescan_sw = "1"b to look for sub-lexemes contained within the list. Sub-lexemes found in the list will be treated as child tokens of the list lexeme.

Four list kinds are supported: square-bracketed parenthesized braced angle-bracketed

square-bracketed-list lexeme

is a list of characters starting with a left-bracket ([) character and ending with the first subsequent right-bracket (]) character. All characters between and including the start and end brackets are treated as part of the lexeme.

A square-bracketed-list may not be nested within an outer square-bracketed-list. It may include several left-bracket characters but ends with the first right-bracket character encountered by the scan. Example: [x + 42 = 50]

nested-list lexemes

If a list begins with one of the nested-list starting characters, then it ends as follows.

- Set nesting count to 1.
- Scan for either the start character of this list kind or its list end character.
    - If another start character is found, increase the nesting count by 1.
    - If the list's end character is found, decrease the nesting count by 1, and continue scanning for end-of-list until the nesting count equals 0.

This permits a nested list to include any character. Both its starting and ending character are included in the lexeme.

The three nested-list kinds are as follows.

parenthesized-list lexeme

a list of characters starting with a left-parenthesis (() and ending with a right-parenthesis ()) character. Examples: (a + b) (45.2) (5 + (a * b) - c)

braced-list lexeme

a list of characters starting with a left-brace ({) and ending with a right-brace (}).
Example: {abc {def ghi} jkl}

angle-bracketed-list lexeme

a list of characters starting with a less-than (<) and ending with a greater-than (>) character.
Example: <statement-kinds>

Proposed Changes

The most significant changes are:

- Adding the bit_string (bit) command/active-function.
- Adding the virtual_pointer (vptr) command/active-function.
- Enhancing cv_ptr_ to include all the methods for locating segments known to:
  - dump_segment and its ring_zero_dump entry point;
  - the existing cv_ptr_ function; and
  - the object_lib_ subsystem for locating named entry points in object segments and object MSFs.

Public hardcore segments in the file system, and some wired data segments can be referenced by name if the user has access to either the >sss>metering_gate_ or the >sl1>phcs_ gate.

The complete set of virtual pointer formats accepted by the new cv_ptr_ implementation is summarized below.

```
2025-10-10  Virtual Pointers

List of virtual pointer components (segment identifier):
 segno   ref_name path

List of virtual pointer components (offset):
 entry_pt   W   (B)   [R]

List of virtual pointer formats:
 segno|W(B)[R]                      ref_name$
 segno|W(B)                          -name ref_name$..., -nm ref_name$...
 segno|W                            path|W(B)[R]
 segno|, segno                      path|W(B)
 segno|entry_pt[R]                  path|W
 segno|entry_pt                     path|, path
 -name segno|..., -nm segno|...     path|entry_pt[R]
 ref_name$entry_pt[R]               path|entry_pt
 ref_name$entry_pt                  dir>entry$entry_pt
 ref_name$W(B)[R]                   <dir>entry$entry_pt
 ref_name$W(B)                      <entry$entry_pt
 ref_name$W

List of null pointer formats:
 -1|1   77777|1   777777|1   null, null()
 -1     77777     777777
```

Use of the new -name control argument forces the segment identifier to be treated as a pathname, even if it has the characteristics of an octal segment number or a reference name.

The [R] component in square brackets displays the ring number stored in a real pointer. By default, the ring number is displayed if the pointer references a ring other than the current ring of execution. A ring number component may now be given in a VIRTUAL_PTR string to ask cv_ptr_ to create a pointer for use in a ring equal to or higher than the current ring of execution. [1]

[1] cv_ptr_ will try to return a pointer targeted for a ring lower than the current ring of execution. But its function return value gets assigned to a pointer stored in the current ring of execution. The Multics pointer store instruction (SPRI) adjusts the ring number stored in that pointer to the larger of: the current ring of execution; or the ring number being returned by cv_ptr_.

Changes by Ticket:

```
Ticket_373 - bit_string version 1.0:
  B-1) Add new bit_string (bit) command/AF to bound_full_cp_ (where
       string.pl1 lives).
  B-2) Add new bit_string.info (bit.info) segment to document the new
       command/AF in sss.info.
  B-3) In ssu_request_tables_.alm, add bit_string (bit) to
       ssu_ exec_com_toolkit request list.
         A) Add it as a multics_request to ssu_request_tables_.alm
            exec_com_toolkit request table so it becomes available in
            exec_coms for azm and other ssu_ subsystems.
         B) Add a link to >doc>info>bit_string.info segment in the
            >doc>ss>ssu_info_dirs>exec_com_toolkit directory so help for
            bit_string is available to azm and other ssu_ subsystems that
            use ssu_-style help requests.
  B-4) Add a new ssu_parm_program.incl.pl1 segment which contains
       declarations and procedures useful in new commands, active functions,
       requests, and active requests.

 Ticket_389 - vptr & cv_ptr_ version 2.0:
  P-1) Replace cv_ptr_.pl1 in bound_conversion_rtns_ with a new
       implementation called cv_ptr_.rd to integrate its code with a new
       virtual_pointer (vptr) command/AF that provides an active
       function interface for interpreting virtual_pointers.  Both
       cv_ptr_ and vptr share the same code to parse virtual_pointers.
         A) The version associated with this combining of cv_ptr_ &
            virtual_ptr will be known as:  virtual_ptr 2.0
         B) Replace cv_ptr_.info to describe changes to the cv_ptr function.
  P-2) Add virtual_pointer.info (vptr.info) to the sss.info library.  It
       documents vptr as a command/AF and as a request usable in ssu_
       subsystems.
  P-3) Change virtual_pointers.gi.info to describe the ring number field
       of a pointer and its representation in a VIRTUAL_PTR string.
  P-4) Add a new cv_ptr_$details entry point which works like the cv_ptr_
       entry point, but also returns to the caller with data describing
       the located segment.
         A) Add cv_ptr_$details entry point.
         B) Add cv_ptr_details.incl.pl1 include segment declaring a
            structure holding the returned data.
         C) Modify cv_ptr_.info to describe this new entry point along
            with details of the data returned about the located segment.
  P-5) In ssu_request_tables_.alm, add virtual_pointer (vptr) to the ssu_
       exec_com_toolkit request list.
         A) Add it as a multics_request to ssu_request_tables_.alm
            exec_com_toolkit request table so it becomes available in
            exec_coms for azm and other ssu_ subsystems.
         B) Add a link to >doc>info>virtual_ptr.info segment in the
            >doc>ss>ssu_info_dirs>exec_com_toolkit directory so help for
            virtual_pointer (vptr) is available to azm and other ssu_
            subsystems that use ssu_-style help requests.
  P-6) vptr also uses the new ssu_parm_program.incl.pl1 segment which
       contains declarations and procedures useful in new commands and
       active functions.
```

```
 Ticket_375
  DS_1) Modify dump_segment.pl1 to use cv_ptr_$details to obtain a pointer
        to its data to be dumped (by dump_segment and ring_zero_dump
        commands/AFs).
  DS-2) Modify dump_segment argument processing to make use of the new
        ssu_parm_program.incl.pl1 segment which contains declarations and
        procedures useful in new commands and active functions.
  DS-3) Give correct gate names in ring_zero_peek_.info, which is called
        by dump_segment.pl1 to access inner-ring data.


In bound_ssu_:
 Ticket_377
  S-1)  Add new ssu_tokens_.pl1 to ssu_ to be called by the new
        virtual_pointer (vptr) command/AF.
          - new entrypoints:
             ssu_$display_tokens, ssu_$get_tokens, ssu_$token_id,
             ssu_$find_token_with_id, ssu_$get_token_area
          A) Add the new entry points to ssu_.alm.
          B) Add a new ssu_tokens_.pl1 subroutine to bound_ssu_.
          C) Add a new ssu_token.incl.pl1 include segment to declare the
             slightly-modified token format accepted by reductions-based
             translators.
          D) Add declarations for the new ssu_ entry points to:
               >sss>pl1.dcl


 Ticket_325
  S-2) In _ssu_check_sci.incl.pl1, clarify messages describing
       any problems it finds with the ssu_ sci_ptr argument.
         A) Recompile all programs which include this include segment.
             ----------------------------------------
            pcref _ssu_check_sci.incl.pl1
            References to _ssu_check_sci.incl.pl1:  (08/04/87  1140.5 pdt Tue)
                ssu_arglist_.pl1, ssu_execute_.pl1, ssu_invocation_.pl1,
                ssu_misc_procs_.pl1, ssu_misc_requests_.pl1, ssu_procedure_mgr_.pl1,
                ssu_request_mgr_.pl1, ssu_request_processor_.pl1, ssu_temp_mgr_.pl1,
                ssu_usage_.pl1

            References to _ssu_check_sci.incl.pl1:  (08/06/87  0913.4 pdt Thu)
                sc_get_error_name_.pl1, ssu_ec_.pl1, ssu_error_.pl1,
                ssu_info_mgr_.pl1, ssu_listen_.pl1

         B) Change ssu_ec_.pl1 and ssu_error_.pl1 to copy the sci_ptr
            argument only after its value has been checked by this
            include segment.  Copying before testing may fail, since
            one of the include checks is for a correct modifier field in
            the sci_ptr ITS pointer.  Copying fails if the modifier is
            not valid.
```

```
Ticket_376
  S-3) Change ssu_.info to document the new and forgotten ssu_
       entry points.
  S-4) Change >sss>pl1.dcl to include declarations for new and
       forgotten ssu_ entry points.
        - previously undocumented entrypoints:
            ssu_$invoke_request, ssu_$locate_request
  S-5) Correct entry in pl1.dcl for hcs_$get_user_access_modes_seg.
       It was misnamed hcs_$get_user_access_modes_ptr, an entry point
       that does not exist.
```

Changes to the multics_libraries:

```
Bound_obj:    bound_conversion_rtns_            IN: sss  UPDATE;
  bindfile:   bound_conversion_rtns_.bind        REPLACE;
  source:     cv_ptr_.pl1                        DELETE   compiler: pl1 -ot;
  source:     cv_ptr_.rd                         ADD      compiler: rdc
                                                          -ot -trace off;

Bound_obj:    bound_full_cp_                    IN: sss  UPDATE;
  bindfile:   bound_full_cp_.bind                REPLACE;
  source:     bit_string.pl1                     ADD      compiler: pl1 -ot;

Bound_obj:    bound_misc_commands_              IN: sss  UPDATE;
  bindfile:   bound_misc_commands_.bind          REPLACE;
  source:     dump_segment.pl1                   REPLACE  compiler: pl1 -ot;

Bound_obj:    bound_ssu_                        IN: sss  UPDATE;
  bindfile:   bound_ssu_.bind                    REPLACE;
  source:     ssu_.alm                           REPLACE  compiler: alm;
  source:     ssu_arglist_.pl1                   REPLACE  compiler: pl1 -ot;
  source:     ssu_ec_.pl1                        REPLACE  compiler: pl1 -ot;
  source:     ssu_error_.pl1                     REPLACE  compiler: pl1 -ot;
  source:     ssu_execute_.pl1                   REPLACE  compiler: pl1 -ot;
  source:     ssu_info_mgr_.pl1                  REPLACE  compiler: pl1 -ot;
  source:     ssu_invocation_.pl1                REPLACE  compiler: pl1 -ot;
  source:     ssu_listen_.pl1                    REPLACE  compiler: pl1 -ot;
  source:     ssu_misc_procs_.pl1                REPLACE  compiler: pl1 -ot;
  source:     ssu_misc_requests_.pl1             REPLACE  compiler: pl1 -ot;
  source:     ssu_procedure_mgr_.pl1             REPLACE  compiler: pl1 -ot;
  source:     ssu_request_mgr_.pl1               REPLACE  compiler: pl1 -ot;
  source:     ssu_request_processor_.pl1         REPLACE  compiler: pl1 -ot;
  source:     ssu_request_tables_.alm            REPLACE  compiler: alm;
  source:     ssu_temp_mgr_.pl1                  REPLACE  compiler: pl1 -ot;
  source:     ssu_tokens_.pl1                    ADD      compiler: pl1 -ot;
  source:     ssu_usage_.pl1                     REPLACE  compiler: pl1 -ot;

Bound_obj:    bound_system_control_             IN: hard  UPDATE;
  source:     sc_get_error_name_.pl1             REPLACE  compiler: pl1 -ot;

Include:      _ssu_check_sci.incl.pl1           IN: sss.include  REPLACE;
Include:      cv_ptr_details.incl.pl1           IN: sss.include  ADD;
Include:      ssu_parm_program.incl.pl1         IN: sss.include  ADD;
Include:      ssu_token.incl.pl1                IN: sss.include  ADD;

Info:         bit_string.info                   IN: sss.info  ADD;
               add_name:  bit.info;
Info:         cv_ptr_.info                      IN: sss.info  REPLACE;
Info:         dump_segment.info                 IN: sss.info  REPLACE;
               add_name:  ds.info;
Info:         dump_segment_.info                IN: sss.info  REPLACE;
Info:         ring_zero_dump.info               IN: priv.info REPLACE;
               add_name:  rzd.info;
Info:         ring_zero_peek_.info              IN: sss.info  REPLACE;
Info:         ssu_.info                         IN: sss.info  REPLACE;

Info:         virtual_pointer.info              IN: sss.info  ADD;
```

```
                add_name:  vptr.info;
Info:          virtual_pointers.gi.info          IN: sss.info  REPLACE;
                add_name:  virtual_pointers.info;

Seg(dcl_file): pl1.dcl                           IN: sss.execution  REPLACE;
```

New or Changed Include Segments:

Declarations in the new cv_ptr_details.incl.pl1 include segment are described in the cv_ptr_.info documentation.

Declarations in the new ssu_token.incl.pl1 include segment are described in the ssu_$get_tokens documentation (see the changes to ssu_.info).

Modifications to the existing _ssu_check_sci.incl.pl1 segment are shown below.

```
compare_ascii >ldd>include>_ssu_check_sci.incl.pl1 ==

A9          1) change(87-02-07,GDixon), approve(87-05-25,MCR7680),
A10            audit(87-06-02,Parisek), install(87-08-04,MR12.1-1056):
Changed by B to:
B9          1) change(1987-02-07,GDixon), approve(1987-05-25,MCR7680),
B10            audit(1987-06-02,Parisek), install(1987-08-04,MR12.1-1056):


Inserted in B:
B14         2) change(2023-11-28,GDixon):
B15            Clarify sub_err_ messages to identify which pointer or structure element
B16            has not passed the validation check.
Preceding:
A14                                                     END HISTORY COMMENTS */


A43                     call sub_err_ (error_table_$bad_ptr, SSU_, ACTION_CANT_RESTART,
                                       null (), (0), "^24.3b", unspec (p_sci_ptr) );
Changed by B to:
B46                     call sub_err_ (error_table_$bad_ptr, SSU_, ACTION_CANT_RESTART, null (), (0),
B47                                    " sci_ptr has invalid ITS modifier: ^24.3b",
                                          unspec (p_sci_ptr));


A49                     call sub_err_ (error_table_$null_info_ptr, SSU_, ACTION_CANT_RESTART,
                                          null (), (0), "sci_ptr");
Changed by B to:
B53                     call sub_err_ (error_table_$null_info_ptr, SSU_, ACTION_CANT_RESTART,
                                          null (), (0), " sci_ptr");


A57                 call sub_err_ (error_table_$unimplemented_version, SSU_, ACTION_CANT_RESTART,
                           null (), (0), "^24.3b",
A58                        unspec (p_sci_ptr -> sci.version));
A59                 go to RESIGNAL_BAD_VERSION;
A60         \014
A61         %include its;
A62         \014
Changed by B to:

B61                 call sub_err_ (error_table_$unimplemented_version, SSU_, ACTION_CANT_RESTART,
                                     null (), (0),
B62                                 " sci_ptr -> sci.version: ^24.3b",
                                     unspec (p_sci_ptr -> sci.version));
B63                 go to RESIGNAL_BAD_VERSION;
B64         %page;
B65         %include its;
B66         %page;

Comparison finished: 5 differences, 24 lines.
```

Contents of the new include segment ssu_parm_program.incl.pl1 are shown below. This include segment will be recommended for use more widely in a future MCR describing a tool which automates processing of input arguments for command and active function programs. As a prelude to that development work, this include segment is now included by cv_ptr_.rd and bit_string.pl1 to test the declarations and code procedures in this segment.

```
/*  START OF:        ssu_parm_program_.incl.pl1               *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  * */

          /*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  * */
          /*                                                                                                    */
          /* Declarations and subroutines/functions useful in a Subsystem Utilities (ssu_) programming environment.  */
          /*                                                                                                    */
          /*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  * */

/****^  HISTORY COMMENTS:
  1) change(2024-08-26,GDixon):
     Initial version based upon ssu_standalone_command_.incl.pl1.
  2) change(2025-05-29,GDixon):
     Add declaration for new subroutines:
         ssu_$display_tokens, ssu_$find_token_with_id,
         ssu_$get_tokens, ssu_$get_token_area, and ssu_$token_id.
                                                    END HISTORY COMMENTS */

  dcl  active_fnc_err_ entry options(variable);            /* Command / ssu_ request support routines.             */
  dcl  com_err_ entry() options(variable);
  dcl  cu_$arg_list_ptr entry() returns (ptr);
  dcl  ssu_$abort_line entry() options(variable);
  dcl  ssu_$abort_subsystem entry() options(variable);
  dcl  ssu_$arg_list_ptr entry (ptr, ptr);
  dcl  ssu_$arg_ptr entry (ptr, fixed bin, ptr, fixed bin(21));
  dcl  ssu_$create_invocation entry (char(*), char(*), ptr, ptr, char(*), ptr, fixed bin(35));
  dcl  ssu_$destroy_invocation entry (ptr);
  dcl  ssu_$execute_line entry (ptr, ptr, fixed bin(21), fixed bin(35));
  dcl  ssu_$execute_string entry (ptr, char(*), fixed bin(35));
  dcl  ssu_$get_area entry (ptr, ptr, char(*), ptr);
  dcl  ssu_$get_info_ptr entry (ptr) returns(ptr);
  dcl  ssu_$get_subsystem_name entry (ptr) returns(char(32));
  dcl  ssu_$get_temp_segment entry (ptr, char(*), ptr);
  dcl  ssu_$get_token_area entry (ptr, char(*)) returns(ptr);
  dcl  ssu_$listen entry (ptr, ptr, fixed bin(35));
  dcl  ssu_$print_message entry() options(variable);
  dcl  ssu_$release_area entry (ptr, ptr);
  dcl  ssu_$release_temp_segment entry (ptr, ptr);
  dcl  ssu_$reset_procedure entry (ptr, char(*), fixed bin(35));
  dcl  ssu_$return_arg entry (ptr, fixed bin, bit(1) aligned, ptr, fixed bin(21));
  dcl  ssu_$set_procedure entry (ptr, char(*), entry, fixed bin(35));
  dcl  ssu_$set_prompt entry (ptr, char(64) var);
  dcl  ssu_$set_prompt_mode entry (ptr, bit(*));
  dcl  ssu_$standalone_invocation entry (ptr, char(*), char(*), ptr, entry, fixed bin(35));

  dcl (F init("0"b), T init("1"b)) bit(1) aligned int static options(constant);
                                                           /* Constants used in this include file.                 */

  dcl  ssu_et_$subsystem_aborted fixed bin(35) ext static;  /* External variables.                                 */

  dcl  cleanup condition;                                  /* Conditions used in this include file.                */

  dcl  code fixed bin(35);                                 /* Multics status code.                                 */
%page;

          /*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *      */
          /*                                                                                          */
          /* Support Declarations:  argument processing, and ssu_$standalone_invocation and aborting   */
          /*                                                                                          */
          /*  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *  *      */

  dcl 1 arg_data              aligned,                     /* Data mapped to PARM variable(s)                      */
      2 argI                  fixed bin,                   /*   index of argument mapped to this PARM value, or    */
                                                           /*     0 if no argument was mapped                      */
      2 argL                  fixed bin(21),               /*   length of argument mapped                          */
      2 argP                  ptr;                         /*   pointer to argument mapped                         */

  dcl 1 ctl_data              aligned,                     /* Data mapped to PARM's preceding -ctl_arg             */
      2 ctlI                  fixed bin,                   /*   index of -ctl_arg preceding this PARM operand, or  */
                                                           /*     0 if no argument was mapped to this operand      */
      2 ctlL                  fixed bin(21),               /*   length of -ctl_arg mapped                          */
      2 ctlP                  ptr;                         /*   pointer to -ctl_arg mapped                         */
```

```
  dcl 1 initiate_file_$component_data
                            aligned,
                                                  /*  kind:  archive_pathname                          */
      2 dir                 char(168) unal,       /*   dirname part of archive_pathname                */
      2 ent                 char(32) unal,        /*   entryname part of archive_pathname              */
      2 comp                char(32) unal,        /*   component part of archive_pathname              */
                                                  /*  service:  initiate_file_$component               */
      2 mode                bit(3) unal,          /*   minimum access mode needed (bits: REW)          */
      2 P                   ptr,                  /*   pointer to initiated segment                    */
      2 bc                  fixed bin(24),        /*   bit count of initiated segment                  */
      2 L                   fixed bin(21);        /*   character length of initiated segment           */

%page;

  dcl 1 status_long_data    aligned,
      2 entry_type          fixed bin (2) unaligned unsigned,
                                                  /* 0: link  1: segment  2: directory  3: multi-seg file   */
      2 names_N             fixed bin (16) unaligned unsigned,
                                                  /* number of names                                  */
      2 names_relp          bit (18) unaligned,   /* see entry_names declaration below                */
      2 dtcm                bit (36) unaligned,   /* date/time contents last modified                 */
      2 dtu                 bit (36) unaligned,   /* date/time last used                              */
      2 effective_mode      bit (5) unaligned,    /* caller's effective access                        */
      2 raw_mode            bit (5) unaligned,    /* caller's raw "rew" modes                         */
      2 pad1                bit (8) unaligned,
      2 records_used        fixed bin (18) unaligned unsigned,
                                                  /* number of NONZERO pages used                     */
                                                  /*  -- Limit of information returned by hcs_$status_ --   */

      2 dtd                 bit (36) unaligned,   /* date/time last dumped                            */
      2 dtem                bit (36) unaligned,   /* date/time branch last modified                   */
      2 lvid                bit (36) unaligned,   /* logical volume ID                                */
      2 length_in_pages     fixed bin (12) unaligned unsigned,
                                                  /* number of last page used                         */
      2 bit_count           fixed bin (24) unaligned unsigned,
                                                  /* reported length in bits                          */
      2 pad2                bit (8) unaligned,
      2 copy_switch         bit (1) unaligned,    /* copy switch                                      */
      2 tpd_switch          bit (1) unaligned,    /* transparent to paging device switch              */
      2 mdir_switch         bit (1) unaligned,    /* is a master dir                                  */
      2 damaged_switch      bit (1) unaligned,    /* salvager warned of possible damage               */
      2 synchronized_switch bit (1) unaligned,    /* DM synchronized file                             */
      2 pad3                bit (5) unaligned,
      2 ring_brackets       (0:2) fixed bin (6) unaligned unsigned,
                                                  /*   Write Bracket:  current_ring <= ring_brackets(0)     */
                                                  /*    Read Bracket:  current_ring <= ring_brackets(1)     */
                                                  /* Execute Bracket:  (see MPM Ref. Guide AG91 page 6-23)  */
                                                  /*    ring_brackets(0:2) are defined for a segment        */
                                                  /*    ring_brackets(0:1) are defined for a directory      */
      2 uid                 bit (36) unaligned,   /* unique ID                                        */
                                                  /*  - Limit of information returned by hcs_$status_long - */

      2 length_in_words     fixed bin(19),        /* length of segment in words   [range:  0 ..  262143]    */
                                                  /*   seg:  .length_in_pages * 1024                      */
      2 length_in_bits      fixed bin(24);        /* length of segment in bits    [range:  0 .. 9437148]    */

  dcl  status_area_ptr      pointer,
       status_entry_names   (status_long_data.names_N) character (32) aligned
                            based (pointer (status_area_ptr, status_long_data.names_relp));

  dcl 1 status_link_data    aligned based(status_link_ptr),
      2 entry_type          fixed bin (2) unaligned unsigned,
                                                  /* 0: link  1: segment  2: directory  3: multi-seg file   */
      2 names_N             fixed bin (16) unaligned unsigned,
                                                  /* number of names                                  */
      2 names_relp          bit (18) unaligned,   /* see entry_names declaration below                */
      2 dtem                bit (36) unaligned,   /* date/time entry last modified                    */
      2 dtd                 bit (36) unaligned,   /* date/time last dump                              */
      2 link_path_length    fixed bin (17) unaligned,     /* see linkpath                             */
      2 link_path_relp      bit (18) unaligned,   /* see linkpath                                     */
       status_link_ptr      pointer;
                                                  /* array of names returned                          */
  dcl  status_pathname      character (status_link_data.link_path_length) aligned
                            based (pointer (status_area_ptr, status_link_data.link_path_relp));
                                                  /* link target path                                 */
%page;
```

```
  dcl 1 ssu_interface         aligned,                    /* INTERFACE variables used in programmer's code          */

                                                          /*  ssu_$standalone_invocation arguments                  */
        3 sciP                ptr,                        /*    ssu_sci_ptr                                         */
        3 argListP            ptr,                        /*    pointer to command's argument list                  */
        3 is_standalone       bit(1) aligned,             /*    T: If this is a ssu_ standalone invocation          */

                                                          /*  ssu_$return_arg data                                  */
        3 argCount            fixed bin,                  /*    count of arguments in this program call             */
        3 is_active_string    bit(1) aligned,             /*    T: If this program invoked via an active-string     */
        3 actStrP             ptr,                        /*       described by this pointer                        */
        3 actStrL             fixed bin(21),              /*       and length                                       */

                                                          /*  get_current_arg and get_operand_arg data  (see below) */
        3 arg_examinedI       fixed bin,                  /*   index of last argument examined                     */
        3 arg_positionalJ     fixed bin,                  /*   number of highest positional argument found         */
        3 arg_data,                                       /*   data for arg_examinedI+1 returned by ssu_$arg_ptr   */
          4 argI              fixed bin,                  /*    index   of current argument                        */
          4 argL              fixed bin(21),              /*    length  of current argument                        */
          4 argP              ptr,                        /*    pointer to current argument                        */

                                                          /*  argument mapping data                                 */
        3 arg_in_errorI       fixed bin;                  /*   index of last argument encountering fatal error     */

  dcl 1 ssu_token_area_data   aligned,                    /*  ssu_$get_token_area data                               */
      2 tokenAreaP            ptr,                        /*   pointer to the extensible, no-free area segment       */
       tokenArea              area based( ssu_token_area_data.tokenAreaP );
                                                          /*   Call ssu_$release_area to release the area segment.  */
%page;
abort_to_EXIT: procedure();                               /* Called by ssu_$abort_line & ssu_$abort_subsystem.      */
               goto EXIT;                                 /* Do non-local transfer, to end command/AF and cleanup   */
                                                          /*   Including program must have a statement with an      */
                                                          /*   EXIT label.                                          */

arg_setup:     entry( Assu );

  dcl 1 Assu                  aligned like ssu_interface;

               call ssu_$return_arg ( Assu.sciP, Assu.argCount, Assu.is_active_string, Assu.actStrP, Assu.actStrL );
               Assu.argP = null();
               return;


args_remain:   entry( Assu ) returns ( bit(1) aligned );  /* Return T if input arguments remain to be examined.     */
                                                          /*        F otherwise.                                    */

               return ( Assu.arg_examinedI < Assu.argCount );


fatal_error:   entry( Assu );                             /* Record arg index of most recent argument triggering a  */
                                                          /*   fatal error (one that causes program to abort).      */
               Assu.arg_in_errorI = Assu.arg_examinedI + 1;

               if  F  then do;                            /* Avoid PL1 warnings about procedures not referenced.    */
  dcl  dummy_entry             entry options(variable) variable;
                    dummy_entry = get_current_arg;
                    dummy_entry = assign_arg;
                    dummy_entry = isControlArg;
                    end;
               return;


get_current_arg:                                          /* Access current argument being examined.                */
               entry( Assu, Aarg );

  dcl 1 Aarg                  aligned like arg_data;      /* Sets either arg_data or ctl_data structure elements.   */
  dcl  arg                    char(Aarg.argL) based(Aarg.argP);

GET_CURRENT_ARG:
               if  Assu.arg_examinedI + 1 <= Assu.argCount  then do;
                    Aarg.argI = Assu.arg_examinedI + 1;
                    call ssu_$arg_ptr( Assu.sciP, Aarg.argI, Aarg.argP, Aarg.argL );
                    end;
               return;
```

```
get_next_arg:  entry( Assu, Aarg ) returns( bit(1) aligned );
                                                  /* If next argument to be examined exists in arg_list ... */
                                                  /*   make it current arg being examined  and return (T)   */
               if  Assu.arg_examinedI + 1 < Assu.argCount  then do;
                    Assu.arg_examinedI = Assu.arg_examinedI + 1;
                                                  /* Treat previous argument as having been processed.      */
                    Aarg.argI = Assu.arg_examinedI + 1;     /*   Access next argument in arg list as the operand...   */
                    call ssu_$arg_ptr( Assu.sciP, Aarg.argI, Aarg.argP, Aarg.argL );
                    return (T);
                    end;
               Aarg.argI, Aarg.argL = 0;          /* Oops, there is no next argument on the arg list.       */
               Aarg.argP = null();
               return (F);


get_operand_arg:
               entry( Assu, Aarg ) returns( bit(1) aligned );
                                                  /* If next argument to be examined exists in arg_list     */
                                                  /* and it is NOT a -control_arg ...                       */
                                                  /*   make it current arg being examined  and return (T)   */

               if  Assu.arg_examinedI + 1 < Assu.argCount  then do;
                    Assu.arg_examinedI = Assu.arg_examinedI + 1;
                                                  /* For now, treat previous arg as having been processed.  */
                    Aarg.argI = Assu.arg_examinedI + 1;
                    call ssu_$arg_ptr( Assu.sciP, Aarg.argI, Aarg.argP, Aarg.argL );

                    if  isControlArg( arg )  then do;       /* If next arg looks like -ctl_arg, reject it as an       */
                                                  /*   operand.  Negative number (-23_o) is NOT treated     */
                                                  /*   as a -ctl_arg.                                       */
                         Assu.arg_examinedI = Assu.arg_examinedI - 1;
                                                  /*   Undo treating previous arg as having been processed. */
                         return (F);              /*   Tell caller no operand was found.                    */
                         end;
                    return (T);                   /* Tell caller arg now contains the operand.              */
                    end;
               Aarg.argI, Aarg.argL = 0;          /* Oops, arg_list has no additional arguments.            */
               Aarg.argP = null();
               return (F);


return_current_arg:                               /* Return arg acquired via get_next_arg or get_operand    */
               entry( Assu, Aarg );               /*   to the unexamined argument pool                      */

  dcl  max                  builtin;

               if  Aarg.argI > 0  &  Aarg.argI <= Assu.argCount  then do;
                    Assu.arg_examinedI = max( Assu.arg_examinedI - 1, 0 );
                    goto GET_CURRENT_ARG;
                    end;
               return;


standalone_cleanup_handler:
               entry( Assu );

               if  Assu.is_standalone then do;
                    call ssu_$destroy_invocation( Assu.sciP );
                    Assu.is_standalone = F;
                    end;
               return;

               end abort_to_EXIT;
%page;
```

```
assign_arg:
          proc( Assu, Aarg_data );

  dcl 1 Assu                  aligned like ssu_interface;
  dcl 1 Aarg_data             aligned like arg_data;

          Aarg_data = Assu.arg_data;
          return;

unassign_arg:
          entry( Aarg_data );

          Aarg_data.argL = 0;                             /* Remove data for current argument from PARM.           */
          Aarg_data.argP = null();
          return;

unassign_arg_to_override:
          entry( Aarg_data );

          Aarg_data.argI = 0;                             /* Indicate this PARM has never been assigned.           */
          Aarg_data.argL = 0;                             /* Remove data for current argument from PARM.           */
          Aarg_data.argP = null();
          return;

          end assign_arg;
%page;

isControlArg:                                             /* Returns T if arg is in form of a -control_arg         */
          proc( Aarg ) returns( bit(1) aligned );

  dcl  Aarg                   char(*) unal;
  dcl  argFirst               char(1) defined(Aarg);
  dcl  argSecond              char(1) defined(Aarg) position(2);
  dcl (length, maxlength, reverse, verify
       )                      builtin;

  dcl  ALPHANUMERIC           char(63) int static options(constant)
                              init("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_");


          if  length(Aarg) <= 1 then  goto REJECT;        /* neither "" nor "-" is a -ctl_arg.                     */
          if  argFirst ^= "-"   then  goto REJECT;        /* string not starting with -  is not a -ctl_arg.        */
          if  argSecond = "_"   then  goto REJECT;        /* string starting "-_" is not a -ctl_arg.               */
          if  length(Aarg) > 32 then  goto REJECT;        /* string longer than 32 chars is not a -ctl_arg.        */
          if  verify(reverse(Aarg), ALPHANUMERIC) ^= length(Aarg)  then goto REJECT;
                                                          /* string with non-alphanumeric chars other than leading */
                                                          /*   "-" is not a -ctl_arg.                              */

  dcl  cv_dec_check_ entry (char(*), fixed bin(35)) returns(fixed bin(35));
  dcl (num_check_code, value)  fixed bin(35);

          value = cv_dec_check_( Aarg, num_check_code );  /* If Aarg represents negatively-signed decimal integer  */
          if  num_check_code = 0  then  goto REJECT;      /*   or numeric string with radix indicator, then        */
                                                          /*   it is not a -ctl_arg.                               */
ACCEPT:   return( T );
REJECT:   return( F );


isIdentifier:                                             /* Does string match rules of PL1 <identifier> token?    */
          entry (Aarg) returns( bit(1) aligned );

  dcl  IDENTIFIER             char(64) int static options(constant)
                              init("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789_$");
  dcl  IDENTIFIERfirst        char(52) int static options(constant)
                              init("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz");

  dcl  variableName           char(256) var int static options(constant) init( "" );

          if  length(Aarg) < 1                            then  goto REJECT;
          if  length(Aarg) > maxlength(variableName)      then  goto REJECT;
          if  (verify( argFirst, IDENTIFIERfirst ) ^= 0)  then  goto REJECT;
          if  (verify( Aarg,     IDENTIFIER)       ^= 0)  then  goto REJECT;
          goto ACCEPT;

          end isControlArg;

/*  END OF:         ssu_parm_program_.incl.pl1               * * * * * * * * * * * * * * * * * * * * */
```

Testing

This section of the MCR describes testing performed on the new or changed programs in the proposal. This includes methods of testing, and results obtained from these tests.

Testing the cv_ptr_ Changes

The largest change in this proposal involves revising cv_ptr_.pl1 code. Past attempts at exec_com scripts used the dump_segment (ds) command/active-function (or its inner-ring entry point: ring_zero_dump (rzd)) to read and display data segments. But those scripts needed a more direct command/active-function interface to cv_ptr_ functionality.

A virtual_ptr (vptr) command/AF interface was designed. It would accept a VIRTUAL_PTR string naming a segment to display; then call cv_ptr_ to make the segment known to the user process; and vptr would return the result as a simplified VIRTUAL_PTR string in SEGNO|OFFSET format.

Because cv_ptr_ and virtual_ptr command/AF were both parsing the VIRTUAL_PTR input string, the vptr design was changed to add virtual_ptr (vptr) as an entry point in cv_ptr_.pl1. Then virtual_ptr would have direct access to the details cv_ptr_ learns as it initiates the selected segment.

Then dump_segment.pl1 was found to include a very old cv_ptr_ snapshot hidden in its code. The plan was changed again to integrate any features of that old version missing from the external cv_ptr_ into the revised cv_ptr_.pl1; then have dump_segment call the external cv_ptr_. The cv_ptr_ snapshot inside dump_segment had not been updated for many years and was missing many features of the external cv_ptr_.pl1. Thus, those features were not available to dump_segment users.

- cv_ptr_ snapshot did not support archive pathnames and had difficulties in accessing hardcore segments.
- It knew nothing about using the newer object_lib_ routines to access entry points in the latest versions of object segments.
- It had several bugs in the way it processed its VIRTUAL_PTR string argument.
- But the internal cv_ptr_ knew several tricks for accessing data segments in ways that had not been introduced to the external Multics cv_ptr_ function.

Adding new features to cv_ptr_.pl1 greatly increased complexity of the existing code. The best solution was to completely rewrite cv_ptr_ using external primitives to perform more mundane parts of its work.

- Use an ssu_ standalone invocation to simplify reporting errors and managing resources used by cv_ptr_.
- Use ssu_ token routines to split the input VIRTUAL_PTR string into meaningful lexemes:
  <SEG_ID> <DELIMITER> <OFFSET>
- Use reductions to identify all valid combinations of those lexemes; and then assign meanings to those lexemes.

To simplify cv_ptr_ code, cv_ptr_.pl1 will now be replaced by a cv_ptr_.rd reduction translator program that no longer has to find lexemes, figure out their meaning, etc. Its code focuses on steps needed to construct a real pointer to a selected segment location.

- Locate the target segment.
- Determine attributes (ring location, user's effective access; length) of target.
- Make target known to the user process, if needed.
- Locate any specified offset within the segment.
- Diagnose errors found during these steps.
- Return to the caller with a pointer to the target.

Testing the rewritten code required development of a test suite covering existing and new features. The suite compares outputs of the old and new cv_ptr_ implementation and ensures that any differences are planned and appropriate.

The test suite is called test_cv_ptr (tcp). This testing program won't be installed, but its results are documented below to show the tests that were performed and results found.

The testing was a great success. Many problems in the old code were corrected in the new code. Many bugs in the new cv_ptr_ implementation were found and fixed. All tests now succeed; or if they fail, cause of failure was verified as stemming from the old cv_ptr_, while ensuring the new cv_ptr_ properly handled the failing test case.

```
test_cv_ptr
     The test_cv_ptr (tcp) command compares outputs from the two cv_ptr_ versions
     and displays any differences in returned pointer or status code.
     Each VPTR: line shows:
       - the VIRTUAL_PTR argument for a cv_ptr_ invocation; followed by
       - an Expect: value describing expected components of resulting pointer; and
       - error_table_ values from Old / New cv_ptr_ calls if either is expected
         to fail.
     If "OK" begins the test line, all the expected results were found from both
     cv_ptr_ versions.  Otherwise, difference-from-Expected lines are displayed.

       Note:  To test inner-ring functionality, the user running the tcp command must
              have access to both >sss>metering_gate_ (aka metering_ring_zero_peek_)
              and >sl1>phcs_ gate.  Check your access before running this test.

       Note:  To test use of linker search paths in finding segments, the user's
              linker search paths should begin with the following rules:
                 linker:
                    -initiated_segments
                    -referencing_dir
                    -working_dir
              followed by the normal system library search paths.  The tcp test
              program must reside in a directory immediately below the build
              directory which contains the new cv_ptr_ program.

     TEST SETUP VERIFICATION:  Display segnos of cv_ptr_ versions in Expect data.

       Old cv_ptr_: Installed in >sss Library.
       New cv_ptr_: New cv_ptr_ segment resides in directory immediately above
                    dir containing tcp.pl1.  Test accesses new cv_ptr_  using the
                    reference name:  ncv_.

 OK  VPTR: >sss>cv_ptr_|0                        Expect: 422|0(0)[4]
 OK  VPTR: <cv_ptr_|0                            Expect: 414|0(0)[4]
 OK  VPTR: ncv_$cv_ptr_                          Expect: 414|7504(0)[4]
```

```
    FYI: A return ptr from Old/New cv_ptr_ function always gets [RINGNO] = current_ring
      when it is assigned to the function's return value.  Two exceptions to this rule:
       - when returning a null() pointer value, or
       - when the VIRTUAL_PTR's requested [RINGNO] > current_ring.
      One might expect hardcore segment pointers returned by cv_ptr_ to have [RINGNO] = 0.
      They are always >= current_ring (or 0 if null pointer), which stems from PL1's
      function return(ptr) mechanism.  Old cv_ptr_ does not handle [RINGNO] in its input:
      it returns error_table_$zero_length_seg for some reason.  This expected outcome is
      shown in the 2nd test line below.

                                                                  cv_ptr_ OLD              / NEW  return code
OK  VPTR: 0|0                                 Expect: 0|0(0)[4]
OK  VPTR: 0|0[0]                              Expect: 0|0(0)[4]     et$ zero_length_seg    /


    FYI: Display actual [RINGNO] generated by use of PL1 baseptr builtin.
      Old cv_ptr_ fails when trying to generate a pointer to most ring-0 segments.
      New cv_ptr_ usually sets [RINGNO] = current_ring (with above-noted exceptions).

                                                                  cv_ptr_ OLD              / NEW  return code
OK  PL1 builtin:  baseptr(71)                 Expect: 71|0(0)[4]    et$ no_info            /


    FYI: Display components generated by PL1 null() builtin; [RINGNO] is always set to 0.
      I suspect that PL1 special-cases contents of that value so all null pointers have
      identical content in every ring of execution.  Old cv_ptr_ does not handle "null".

OK  PL1 builtin:  null                        Expect: 77777|1(0)[0]     et$ noentry            /


    PLANNED DIFFERENCE:  Old cv_ptr_ fails at trying to return hardcore segment pointers
      except for segment 0 (dseg) and some other segments.  It returns an error code
      with a null() pointer value for segment 71 (pds) because it incorrectly calls
      object_lib_$initiate for a ring-0 segment.  New cv_ptr_ handles hardcore segments
      more reliably by using several ring0_get_ functions (available to all users) and
      by remembering to special-case hardcore (ring-0) segments thruout the code.

                                                                  cv_ptr_ OLD              / NEW  return code
OK  VPTR: 0                                   Expect: 0|0(0)[4]
OK  VPTR: 71|550                              Expect: 71|550(0)[4]      et$ no_info            /
OK  VPTR: pds$first_covert_event_time         Expect: 71|550(0)[4]      et$ no_info            /


    FYI: Hardcore segnos are from 0 to 227  (dseg  TO  first stack segno - 1).
      Though first stack (segno 230) resides in ring-0, each process has its own
      ring-0 stack (one of the ring-0 stack segments: >sl1>stack_0.nnn) which is
      not described by ring0_get_ hardcore data.  This is not a segment in [pd]
      because the process must run on ring-0 stack when the process first starts
      (before [pd] gets created by that new process).

      When Old cv_ptr_ attempts to initiate segment 230, it fails by returning a null
      pointer with error_table_$invalidsegno because the inner-ring stack is not known
      in the current ring.  New cv_ptr_ special-cases segno 230 because every process
      has a stack_0 in ring-0 even if ring0_get_$name has not been informed of that
      per-process segment.

      Old cv_ptr_ reports lack of access to the other lower-ring stack segments.
      New cv_ptr_ returns the correct pointer (with [RINGNO] = 4 as explained above).
      Dealing with inner-ring pointers is needed so inner-ring data locations may be
      accessed by ring_zero_peek_, ring_zero_dump (rzd) if user has phcs_ access.

                                                                  cv_ptr_ OLD              / NEW  return code
OK  VPTR: 230|1200                            Expect: 230|1200(0)[4]    et$ invalidsegno       /
OK  VPTR: 231|1200                            Expect: 231|1200(0)[4]    et$ moderr             /
OK  VPTR: 232|1200                            Expect: 232|1200(0)[4]    et$ moderr             /


    FYI: 77774 is outside the range of acceptable segnos (those known to any process).
         This check on segno range rules out pathname ending with all-octal entryname
         unless that pathname is enclosed in quotes:  vptr "10000"|40
         would look for a segment with name 10000 using linker search paths.
         So would the input:  vptr -name 10000|40

OK  VPTR: 10000|40                            Expect: 77777|1(0)[0]     et$ invalidsegno       / invalidsegno
OK  VPTR: "10000"|40                          Expect: 77777|1(0)[0]     et$ noentry            / noentry
OK  VPTR: -name 10000|40                      Expect: 77777|1(0)[0]     et$ noentry            / noentry


    Mid-range segnos like 234 (the stack_4 segment) are handled OK by both versions of cv_ptr_.

OK  VPTR: 234|1200(18)                        Expect: 234|1200(18)[4]


    Non-octal segnos like 239 are handled OK by both versions of cv_ptr_ as a pathname
    whose entryname is not found.  Old cv_ptr_ looks only in the working directory.
    New cv_ptr_ searches for entryname 239 using linker search paths.
    Typical error codes are returned for bad pathnames or missing entry.

OK  VPTR: 239|1200(18)                        Expect: 77777|1(0)[0]     et$ noentry            / noentry
OK  VPTR: <>234|1200(18)                      Expect: 77777|1(0)[0]     et$ badpath            / badpath
OK  VPTR: <234|1200(18)                       Expect: 77777|1(0)[0]     et$ noentry            / noentry
OK  VPTR: longentry_01234567890123456789012   Expect: 77777|1(0)[0]     et$ entlong            / entlong


    Test various forms of segno pointer representations.
```

```
      Old cv_ptr_ does not like SP chars starting VIRTUAL_PTR.  In the 1st test below,
      ending SP chars are rtrimmed from the end of an empty string so no error occurs.
      In the 2nd test below, leading SP chars are treated as the entryname part of a
      pathname (same as strange pathname "<  234").  Remaining tests below show
      application of default WORDNO & BITNO values for missing fields in VPTR.

 OK  VPTR: "  "                                  Expect: 0|0(0)[4]
 OK  VPTR: "  234"                               Expect: 234|0(0)[4]        et$ noentry             /
 OK  VPTR: |                                    Expect: 0|0(0)[4]
 OK  VPTR: 234                                  Expect: 234|0(0)[4]
 OK  VPTR: 234|                                 Expect: 234|0(0)[4]
 OK  VPTR: 234|0                                Expect: 234|0(0)[4]
 OK  VPTR: 234|0(0)                             Expect: 234|0(0)[4]

    PLANNED DIFFERENCE:  Old cv_ptr_ never tried to handle [RINGNO] in ptr representation.  It
      reports a variety of different error codes when VIRTUAL_PTR includes [RINGNO] construct.
      New cv_ptr_ handles these correctly.

 OK  VPTR: 234|1200[4]                          Expect: 234|1200(0)[4]     et$ zero_length_seg     /
 OK  VPTR: 234|0(0)[0]                          Expect: 234|0(0)[4]        et$ bad_conversion      /
 OK  VPTR: 234|0(0)[1]                          Expect: 234|0(0)[4]        et$ bad_conversion      /
 OK  VPTR: 234|0(0)[4]                          Expect: 234|0(0)[4]        et$ bad_conversion      /
 OK  VPTR: 234|0(0)[7]                          Expect: 234|0(0)[7]        et$ bad_conversion      /
----------
 OK  VPTR: -1|1[0]                              Expect: 77777|1(0)[0]      et$ improper_data_format /
 OK  VPTR: -1|1[1]                              Expect: 77777|1(0)[0]      et$ improper_data_format /
 OK  VPTR: -1|1[4]                              Expect: 77777|1(0)[0]      et$ improper_data_format /


    PLANNED DIFFERENCE:  Old cv_ptr_ never accepted the string "null" in any form, instead
      treating it as a pathname in working_dir which was not found.  It returned with an
      error code and an actual PL1 null builtin instead of completing the conversion.
      New cv_ptr_ always returns the standard null() builtin value, with [RINGNO] = 0.

 OK  VPTR: null()                               Expect: 77777|1(0)[0]      et$ noentry             /
 OK  VPTR: " null() "                           Expect: 77777|1(0)[0]      et$ noentry             /
 OK  VPTR: " null  (   ) "                      Expect: 77777|1(0)[0]      et$ noentry             /
 OK  VPTR: null                                 Expect: 77777|1(0)[0]      et$ noentry             /
 OK  VPTR: " null "                             Expect: 77777|1(0)[0]      et$ noentry             /
----------
 OK  VPTR: 77777                                Expect: 77777|1(0)[0]
 OK  VPTR: 77777|                               Expect: 77777|1(0)[0]
 OK  VPTR: 777777                               Expect: 77777|1(0)[0]
 OK  VPTR: 777777|                              Expect: 77777|1(0)[0]
----------
 OK  VPTR: -1                                   Expect: 77777|1(0)[0]
 OK  VPTR: -1|                                  Expect: 77777|1(0)[0]


    BUG: Old cv_ptr_ returns [RINGNO] = current_ring for -1|1 (not recognizing it as null ptr).

    VPTR: -1|1                                  Expect: 77777|1(0)[0]
                  OLD cv_ptr_                   NEW cv_ptr_                EXPECT
          ringN:  4                             0                          0


    VPTR: 77777|1                               Expect: 77777|1(0)[0]
                  OLD cv_ptr_                   NEW cv_ptr_                EXPECT
          ringN:  4                             0                          0


    VPTR: 777777|1                              Expect: 77777|1(0)[0]
                  OLD cv_ptr_                   NEW cv_ptr_                EXPECT
          ringN:  4                             0                          0


    Null segno values with non-standard offsets are handled as regular segnos.  Both
    Old/New cv_ptr_ return the unexpected offset with [RINGNO] = current_ring rather than 0.

----------
 OK  VPTR: -1|0                                 Expect: 77777|0(0)[4]
 OK  VPTR: -1|2                                 Expect: 77777|2(0)[4]
 OK  VPTR: 77777|2                              Expect: 77777|2(0)[4]
 OK  VPTR: 777777|2                             Expect: 77777|2(0)[4]


    Other negative segno values are treated as pathname by old cv_ptr_, but as an
    invalid segment number by new cv_ptr_.  User would have to use -name to have a
    negative segno treated as a pathname.

 OK  VPTR: -2|0                                 Expect: 77777|1(0)[0]      et$ noentry             / invalidsegno
 OK  VPTR: -2|1                                 Expect: 77777|1(0)[0]      et$ noentry             / invalidsegno
 OK  VPTR: -3|1                                 Expect: 77777|1(0)[0]      et$ noentry             / invalidsegno
 OK  VPTR: -name -2|1                           Expect: 77777|1(0)[0]      et$ noentry             / noentry
```

```
     Missing WORDNO is handled ok by both versions of cv_ptr_.  Test boundaries for
     other fields of VIRTUAL_PTR:  non-octal WORDNO; limits on (BITNO); limits on
     [RINGNO] values.

 OK  VPTR: 234|(18)                              Expect: 234|0(18)[4]
 OK  VPTR: 234|9                                 Expect: 77777|1(0)[0]      et$ zero_length_seg      / bad_conversion
 OK  VPTR: 234|9(18)                             Expect: 77777|1(0)[0]      et$ zero_length_seg      / bad_conversion
----------
 OK  VPTR: 234|4(-1)                             Expect: 77777|1(0)[0]      et$ bad_conversion       / bad_conversion
 OK  VPTR: 234|4(0)                              Expect: 234|4(0)[4]
 OK  VPTR: 234|4(35)                             Expect: 234|4(35)[4]
 OK  VPTR: 234|4(36)                             Expect: 77777|1(0)[0]      et$ out_of_bounds        / out_of_bounds
----------
 OK  VPTR: 234|4[-1]                             Expect: 77777|1(0)[0]      et$ zero_length_seg      / bad_conversion
 OK  VPTR: 234|4[0]                              Expect: 234|4(0)[4]        et$ zero_length_seg      /
 OK  VPTR: 234|4[7]                              Expect: 234|4(0)[7]        et$ zero_length_seg      /
 OK  VPTR: 234|4[8]                              Expect: 77777|1(0)[0]      et$ zero_length_seg      / out_of_bounds
----------
 OK  VPTR: 234|4(18)[7]                          Expect: 234|4(18)[7]       et$ bad_conversion       /


     To explore bounds checking on segments, the next tests reference a segment
     known to exist in every process: [pd]>pit.  By default, it has a max_length set at
     261120 words (255 pages) long.  Both Old and New cv_ptr_ can reference the last
     bit on the 255th page of that segment.  But Old cv_ptr_ does not report error for
     a reference to the first word of the 256th page, but only for a reference to the
     second word of that page.  It also reports an out_of_bounds error code rather than
     the more appropriate boundviol error code used by new cv_ptr_.

 OK  VPTR: [pd]>pit|775777(35)                   Expect: 264|775777(35)[4]
 OK  VPTR: [pd]>pit|776000                       Expect: 264|776000(0)[4]   et$                      / boundviol
 OK  VPTR: [pd]>pit|776000(1)                    Expect: 264|776000(1)[4]   et$                      / boundviol
 OK  VPTR: [pd]>pit|776001                       Expect: 264|776001(0)[4]   et$ out_of_bounds        / boundviol

     Fortran supports Very Large Arrays (VLAs) declared larger than a single segment can hold
     (255 pages).  Each VLA may hold up to 2**24 words (the storage in 64 segments, each
     holding 256 pages).  The new cv_ptr_ uses each segment's max_length to check for a
     pointer that references beyond the max_length of that segment (a boundviol event).  The
     test above that checks a reference to WORDNO 261120 (776000_o) in a 255K segment verifies
     that the new cv_ptr_ should detect a reference beyond page 256 of a segment whose
     max_length is set to 256K words.


     Pathname Errors:

      cv_ptr_ detects over-length archive_pathnames:  longer than 202 chars (168 + 2 + 32).

 OK  VPTR: a0123456789012345678901234567890123456789z>a0123456789012345678901234567890123456789z>a0123456789012345678901234567890123456789z>a0123456789
\c0123456789012345678901234567890123456789z>a0123456789012345678901234567890123456789z>a0123456789012345678901234567890123456789z>a0123456789012345678901234567890123456789
\cz>a0123456789012345678901234567890123456789z>a0123456789012345678901234567890123456789z>a0123456789012345678901234567890123456789z
                                                Expect: 77777|1(0)[0]      et$ pathlong             / pathlong

      cv_ptr_ calls expand_pathname_$component which also detects over-length pathnames ...

 OK  VPTR: a0123456789012345678901234567890123456789z>a0123456789012345678901234567890123456789z>a0123456789012345678901234567890123456789z>a0123456789
\c0123456789012345678901234567890123456789z>a0123456789012345678901234567890123456789z>a0123456789012345678901234567890123456789z>a0123456789012345678901234567890123456789
\cz>a0123456789012345678901234567890123456789z>a0123456789012345678901234567890123456789z>def
                                                Expect: 77777|1(0)[0]      et$ pathlong             / pathlong

      cv_ptr_ calls object_lib_$initiate to make the seg known to process w/ no reference name.
      Each test terminates the segment (decrements seg's no_reference_name initiation count)
      at end of each test, and verifies that a segment with zero initiation count is no longer
      known to the process.

 OK  VPTR: tc>empty_seg.test_case                Expect: 433|0(0)[4]        w/ term

      Neither Old or New cv_ptr_ complains about a reference beyond bit_count of empty segment
      because caller may want to extend the segment by referencing beyond its current bit_count.

 OK  VPTR: tc>empty_seg.test_case|(1)            Expect: 433|0(1)[4]        w/ term
```

```
      cv_ptr_ calls initiate_file_$component when initiating an archive component.  Since no
      program but the archive command can modify an archive component, any offset from the start
      of that component must fall within the bit_count of the selected component.  In this test,
      the archive component has zero-length: there is no bit to locate w/in the component.
      New cv_ptr_ correctly detects that any VIRTUAL_PTR will reference beyond bit_count of that
      empty archive component, and reports a bounds violation.

      BUG: Next four tests show that Old cv_ptr_ mishandles offsets w/in an archive component.
           It fails to report word offset (which includes WORDNO value at which component
           starts within the archive + the WORDNO offset given in VPTR) extending beyond the
           component's bit_count.


    VPTR: tc>test::empty_seg.test_case           Expect: 77777|1(0)[0]      et$                      / boundviol
                  OLD cv_ptr_                    NEW cv_ptr_                  EXPECT
Pointers disagree:   433|31[4]                   77777|1[0]                   77777|1(0)[0]
           ringN:   4                            0                            0


    VPTR: tc>test::empty_seg.test_case|          Expect: 77777|1(0)[0]      et$                      / boundviol
                  OLD cv_ptr_                    NEW cv_ptr_                  EXPECT
Pointers disagree:   433|31[4]                   77777|1[0]                   77777|1(0)[0]
           ringN:   4                            0                            0


    VPTR: tc>test::empty_seg.test_case|0         Expect: 77777|1(0)[0]      et$                      / boundviol
                  OLD cv_ptr_                    NEW cv_ptr_                  EXPECT
Pointers disagree:   433|0[4]                    77777|1[0]                   77777|1(0)[0]
           ringN:   4                            0                            0


    VPTR: tc>test::empty_seg.test_case|0(1)      Expect: 77777|1(0)[0]      et$                      / boundviol
                  OLD cv_ptr_                    NEW cv_ptr_                  EXPECT
Pointers disagree:   433|0(1)[4]                 77777|1[0]                   77777|1(0)[0]
           ringN:   4                            0                            0


      cv_ptr_ can reference within bit_count of a non-empty segment no matter where it resides.
      The utility_macros segment is exactly 14000_o words long.  Pointers with offset beyond
      end of segment's bit_count are permitted up to the segment's max_length.  A bounds
      violation is reported beyond that point.

 OK  VPTR: tc>utility_macros|13777(35)           Expect: 435|13777(35)[4]   w/ term
 OK  VPTR: tc>utility_macros|14000               Expect: 435|14000(0)[4]    w/ term
 OK  VPTR: tc>utility_macros|757777(35)          Expect: 435|757777(35)[4]  w/ term

    VPTR: tc>utility_macros|776000               Expect: 77777|1(0)[0]      et$ out_of_bounds        / boundviol
                  OLD cv_ptr_                    NEW cv_ptr_                  EXPECT
Pointers disagree:   435|776000[4]               77777|1[0]                   77777|1(0)[0]
           ringN:   4                            0                            0
            code:   0                                                         error_table_$out_of_bounds
            code:                                error_table_$boundviol       error_table_$boundviol


    VPTR: tc>utility_macros|776000(35)           Expect: 77777|1(0)[0]      et$ out_of_bounds        / boundviol
                  OLD cv_ptr_                    NEW cv_ptr_                  EXPECT
Pointers disagree:   435|776000(35)[4]           77777|1[0]                   77777|1(0)[0]
           ringN:   4                            0                            0
            code:   0                                                         error_table_$out_of_bounds
            code:                                error_table_$boundviol       error_table_$boundviol

 OK  VPTR: tc>utility_macros|776001              Expect: 77777|1(0)[0]      et$ out_of_bounds        / boundviol


      When utility_macros is an archive component, any reference beyond its bit_count
      is reported by New cv_ptr_ as a bounds violation.

 OK  VPTR: tc>test::utility_macros|              Expect: 433|62(0)[4]       w/ term

      BUG: Next test shows bug in Old cv_ptr_ again mishandling offset w/in archive component
           while New cv_ptr_ can reference within it at correct offset.  Old cv_ptr_ does not
           add offset of component within the archive to offset given in VIRTUAL_PTR.


    VPTR: tc>test::utility_macros|13777(35)      Expect: 433|14061(35)[4]   w/ term
                  OLD cv_ptr_                    NEW cv_ptr_                  EXPECT
Pointers disagree:   433|13777(35)[4]            433|14061(35)[4]             433|14061(35)[4]


    VPTR: tc>test::utility_macros|14000          Expect: 77777|1(0)[0]      et$                      / boundviol
                  OLD cv_ptr_                    NEW cv_ptr_                  EXPECT
Pointers disagree:   433|14000[4]                77777|1[0]                   77777|1(0)[0]
           ringN:   4                            0                            0
```

```
    Reference_name Errors:

    A reference name can be any ASCII characters of length 1 to 32 chars.  Empty string is
    not a valid reference name.  If no segment has been initiated with given reference name,
    then the name is treated as a pathname.

 OK  VPTR: <$main                                Expect: 77777|1(0)[0]      et$ dirseg              / dirseg
 OK  VPTR: >$main                                Expect: 77777|1(0)[0]      et$ root                / root

    PLANNED DIFFERENCE:  Old cv_ptr_ treats the next four cases as beginning with a ref_name.
      New cv_ptr_ treats first three cases as missing ref_name because a ref_name cannot be
      an empty string.  Fourth case looks for ref_name of ":" and does not find it, then
      searches for an entryname ":" and does not find it.

 OK  VPTR: $                                     Expect: 77777|1(0)[0]      et$ name_not_found      / badarg
 OK  VPTR: $main                                 Expect: 77777|1(0)[0]      et$ name_not_found      / badarg
 OK  VPTR: $0                                    Expect: 77777|1(0)[0]      et$ name_not_found      / badarg
 OK  VPTR: :$main                                Expect: 77777|1(0)[0]      et$ name_not_found      / noentry

    Now check handling of ref_name with a missing offset, or a valid or invalid offset.

 OK  VPTR: vptr$                                 Expect: 414|0(0)[4]
 OK  VPTR: vptr$cv_ptr_                          Expect: 414|7504(0)[4]
 OK  VPTR: pl1_operators_$alm_call               Expect: 260|30267(0)[4]
 OK  VPTR: pl1_operators_$alm_caal               Expect: 77777|1(0)[0]      et$ no_ext_sym          / no_ext_sym
----------
 OK  VPTR: pl1_operators_$30267                  Expect: 260|30267(0)[4]
 OK  VPTR: pl1_operators_$30267(9)               Expect: 260|30267(9)[4]
 OK  VPTR: pl1_operators_$30267(9)[5]            Expect: 260|30267(9)[5]    et$ bad_conversion      /

    BUG:  Old cv_ptr_ can handle some hardcore reference names, but fails others.
          It often fails on names ending with an entry point name.
          New cv_ptr_ handles all of these cases correctly.

 OK  VPTR: dseg$                                 Expect: 0|0(0)[4]
 OK  VPTR: fault_vector$                         Expect: 4|0(0)[4]
 OK  VPTR: pds$                                  Expect: 71|0(0)[4]         et$ no_info             /
 OK  VPTR: pds$first_covert_event_time           Expect: 71|550(0)[4]       et$ no_info             /
 OK  VPTR: prds$                                 Expect: 72|0(0)[4]         et$ noentry             /
 OK  VPTR: prds$last_recorded_time               Expect: 72|332(0)[4]       et$ noentry             /
 OK  VPTR: prds$last_recorded_tine               Expect: 77777|1(0)[0]      et$ noentry             / no_ext_sym
----------
 OK  VPTR: pds$550                               Expect: 71|550(0)[4]       et$ no_info             /
 OK  VPTR: pds$550(18)                           Expect: 71|550(18)[4]      et$ no_info             /
 OK  VPTR: pds$550(9)[7]                         Expect: 71|550(9)[4]       et$ bad_conversion      /

    FYI:  Most processes seem to initiate >tools>null_entry_ by its reference name.  tcp forces
          that to happen if not already done.  Use that null_entry_$null_entry_ to test initiate
          by ref_name, by pathname using absolute path, by pathname using linker search rules.
          Old cv_ptr_ does not handle quoted strings in SEGNAME component of VIRTUAL_PTR arg
          nor searching for an ENTRYNAME using linker search paths.  Last test case below
          is found by New cv_ptr_ because linker search paths include -initiated_segments rule.

 OK  VPTR: null_entry_$null_entry_               Expect: 275|0(0)[4]
 OK  VPTR: >t>null_entry_$null_entry_            Expect: 275|0(0)[4]
 OK  VPTR: " ""null_entry_"" $null_entry_"       Expect: 275|0(0)[4]        et$ name_not_found      /

----------
 OK  VPTR: >t>null_entry_|null_entry_            Expect: 275|0(0)[4]
 OK  VPTR: null_entry_|null_entry_               Expect: 275|0(0)[4]        et$ noentry             /
```

Testing New virtual_ptr Command

The test_cv_ptr suite also does most of the testing for the new virtual_ptr (vptr) command / active-function. The vptr command branches to the same code used by cv_ptr_ to parse its VIRTUAL_PTR input string, diagnose errors and return a result to the user.

The cv_ptr_ interface can only return an error status code to the caller if a problem is found. Code for diagnosing errors in the VIRTUAL_PTR generates a full, descriptive error message when the vptr entry point is invoked. If the cv_ptr_ entry point was called, only a status code is returned.

Added testing focused on the following.

- Manual tests of each vptr control argument to ensure it works as documented; and that the resulting SEGNO|OFFSET string is displayed or returned correctly.
- Manual tests to generate each possible descriptive error message: checking for proper content in the message. This is based on white-box analysis of cv_ptr_.rd code.
- Addition of a vptr$vptr_debug entry point which enables or disables display of vptr (and cv_ptr_) internal data to aid in debugging. If debugging is enabled, the following data is displayed before displaying the result or returning the result in place of the active string.
    - VIRTUAL_PTR input string and -control_args given in the command.
    - Tokens found in the input string with their token_ID and attributes.
    - Reduction statement which matched the input tokens, followed by the token(s) matching that reduction's specifications.
    - Command result displayed by vptr command. Active function result string is returned at this point but not displayed.
    - Internal attributes of the target selected by the VIRTUAL_PTR string.

This debug data is shown on the next page.

```
vptr$vptr_debug on; vptr stack_4|1200; vptr$vptr_debug off


 VIRTUAL_PTR: "stack_4|1200"  -nonzero_bit

"stack_4|1200;"                                        VPTR STATEMENT 1 ON
LINE   1

   1: stack_4   2: |   3: 1200  | int=1200   4: ;  | endStmt


25       / <any>    |           /                                        / VERT_BAR \
stack_4 |

PATH_BAR  / <path>   |           / pathC   LEX delim  LEX     segI(PATH)        / PATH_NAME\
stack_4 |

44       / <wordn>           ;  / wordNC            LEX(2)  offI(WORD)        / RETURN   \
1200 ;

234|1200


     vp_arg:  "stack_4|1200" -    segI: 3 (PATH    )   offsetI: 4 (WORD    )
      pathC:  stack_4
       refC:
      segNC:
        dir:  >process_dir_dir>!BPbDxmFBBBBBBB
        ent:  stack_4

      delim:  |

        epC:
     wordNC:  1200
      bitNC:
     ringNC:

          P:  234|0   000234400043 000000000000

       segN:  234
      wordN:  1200
       bitN:  0
      ringN:  4

       type:  segment
  bit_count:  0 bits
     length:     31744 words     31 pages    65536 max length (words)
                 76000_o         37_o       200000_o
       mode:  REW
        rbs:  4, 4, 4         effective_mode:  REW
   hardcore:  F
 inner-ring:  F
  need_term:  T
 null_value:  F
```

Testing New ssu_ Token Entry Points

New entry points in ssu_tokens_.pl1 provide an alternative to the existing lex_string_ method of finding lexemes (meaningful substrings) within an input string.

- ssu_$get_tokens splits an input string into lexemes and provides a token structure describing each lexeme. These tokens are linked together by forward and backward pointers. They are linked by parent/child pointers if any token describes an enclosed list (e.g., parenthesized-list, braced-list, square-bracket-list, or angle-bracket-list). Content of that list is described by the child tokens.
- ssu_$display_tokens displays the token tree generated from an input string.
- ssu_$token_id returns an identifier for a token which describes location of the token within the tree.
- ssu_$find_token_with_id locates a particular token within the tree.
- ssu_$get_token_area obtains an ssu_ temporary segment containing an area with ideal attributes for allocating a tree of tokens describing lexemes in a string.

These routines were tested by the new cv_ptr_, which calls ssu_$get_token_area and ssu_$get_tokens to split the VIRTUAL_PTR input string into lexemes (tokens). When vptr$vptr_debug is enabled, cv_ptr_ also calls ssu_$display_tokens to display results returned by ssu_$get_tokens.

Also, a special-purpose test_get_tokens (tgt) command was written to call these same ssu_ token-related routines. It accepts an arbitrary input string, and displays tokens found in the result. This test program was used as needed to ensure that the lexeme finding method described above in “Lexemes Supported by ssu_tokens_.pl1” is correctly implemented.

All of these manual tests have produced the expected results.

Testing dump_segment.pl1

The dump_segment (ds) command/active-function is a complex program with a variety of entry points.

- dump_segment (ds) operates as a command or active function which displays or returns data locations selected in a target segment.
- ring_zero_dump (rzd) performs the same functions as dump_segment. If the user has access to >sss>metering_gate_ or >sl1>phcs_, it uses those gates to copy selected data from a hardcore or inner-ring segment out to the user's ring. It then displays the data or returns data as an active string.
- dump_seg_ is a simple subroutine which can display data pointed to by the caller in some of the formats supported by the dump_segment command.
- dump_segment_ displays data in any of the output formats provided by dump_segment.
- dump_segment_$string accepts an input string and displays select data from that string in any of the formats provided by dump_segment.

To simplify testing of the dump_segment.pl1 module, code for the three subroutine entry points was not modified. This means those subroutines do not have to be tested.

Furthermore, dump_segment shares the same user interface and code path as ring_zero_dump. The two commands differ only when cv_ptr_ returns a pointer to a hardcore or inner-ring segment.

- dump_segment reports an error about inner-ring segments not being accessible.
- ring_zero_dump calls the ring_zero_peek_ subroutine to copy the inner-ring data out to the user ring. It then displays that data following the same code path as dump_segment.

The only specific test of ring_zero_dump functionality verified that the two commands follow the same code path within dump_segment.pl1 code after the inner-ring or hardcore data is copied into the caller's ring of execution.

Both dump_segment and ring_zero_dump call a new cv_ptr_$details entry point that operates like cv_ptr_ but returns details about the segment pointed to by the returned pointer. The cv_ptr_$details entry point copies data from cv_ptr_'s internal data structure into a cv_ptr_details structure (declared in the new cv_ptr_details.incl.pl1 include segment).

The caller of cv_ptr_$details provides storage for the structure and fills in the input section of that structure. It then calls cv_ptr_$details which: follows the cv_ptr_ code path; then fills in the output section of the structure just before it returns to the caller. Both dump_segment and ring_zero_dump use these output details to learn whether the pointer locates a hardcore or inner-ring segment, what ring that segment resides in, whether the user has read access to the segment, how long the segment is (limits how much data dump_segment can display), etc.

A dump_segment$ds_debug entry point was added to dump_segment.pl1. If debugging is enabled via that entry point, dump_segment.pl1 sets the cv_ptr_details.input.display_output flag to "1"b. This causes

cv_ptr_$details to display the data in the cv_ptr_details structure before it returns to the caller. Typical data is shown below when ring_zero_dump is asked to display data from the stack_header for stack_0, stack_1 and stack_4 segs of the user's process. Using data returned by cv_ptr_$details, ring_zero_dump can determine exactly which data can be displayed from each segment. Note that each of these segments has a different maximum length constraint, as well as a different pages_used[2] (current length).

```
dump_segment$ds_debug on;  ring_zero_dump stack_(0 1 4) -as stack_header;  dump_segment$ds_debug off

ring_zero_dump: pertinent elements from returned structure:

cv_ptr_details
 .output
  .vptr_info
   .path_name: stack_0   .delimiter: |

 ptr_value =  230|0   000230000043 000000000000

  .ptr_value_info
   .segN:        230   .wordN: 0  .bitN: 0  .ringN: 0
   .hardcore:    T
   .readable      at  230|0  for 16384 (40000_o) words

  .seg_info
   .dir_name:    >sl1
   .ent_name:    stack_0
   .ent_type:    seg
   .uid:                 000000000000
   .max_length:          16384 words
   .pages_used   0               .records_used:  0
   .raw_mode:    rew            .ring_brackets:  0,0,0
   .effective_mode:   null
   .dtcm:        12/31/00 17:00          .dtd:  12/31/00 17:00
   .dtu:         12/31/00 17:00         .dtem:  12/31/00 17:00

000000
stack_header           @ 363|0
  cpm_data_ptr = 0(0) [invalid], combined_stat_ptr = 0(0) [invalid],
  clr_ptr = 0(0) [invalid], max_lot_size = 0, main_proc_invoked = 0,
  run_unit_depth = 0, cur_lot_size = 0, cpm_enabled = "000000"b3,
  system_free_ptr = 436|0 (Invalid segment number),
  user_free_ptr = 436|0 (Invalid segment number), null_ptr = 0(0) [invalid],
  stack_begin_ptr = 230|100 (Invalid segment number),
  stack_end_ptr = 230|720 (Invalid segment number),
  lot_ptr = 15|0   lot (ring 0), signal_ptr = 40|14710   >sl1>bound_library_1_,
  bar_mode_sp = 0(0) [invalid],
  pl1_operators_ptr = 41|21766   >sl1>bound_library_wired_,
  call_op_ptr = 41|30267   >sl1>bound_library_wired_,
  push_op_ptr = 41|30273   >sl1>bound_library_wired_,
  return_op_ptr = 41|30317   >sl1>bound_library_wired_,
  return_no_pop_op_ptr = 41|30325   >sl1>bound_library_wired_,
  entry_op_ptr = 41|30307   >sl1>bound_library_wired_,
  trans_op_tv_ptr = 0(0) [invalid], isot_ptr = 15|0   lot (ring 0),
  sct_ptr = 25|406     bound_hc_data_wired (ring 0), unwinder_ptr = null,
  sys_link_info_ptr = 0(0) [invalid], rnt_ptr = 0(0) [invalid],
  ect_ptr = 0(0) [invalid], assign_linkage_ptr = 0(0) [invalid],
  heap_header_ptr = null
  trace                @ 363|72
    frames             @ 363|72
      count = 0, top_ptr = 0|0 [invalid]
    in_trace = "000000000000"b3
  OFF: have_static_vlas

ring_zero_dump: pertinent elements from returned structure:
```

[2] For hardcore segments, pages_used, dates, uid and other information cannot be determined since hcs_$status_long cannot be called for most hardcore segments. Most have no branch in the file system. max_length is obtained from the Segment Descriptor Word (SDW) bound field of a hardcore segment.

```
cv_ptr_details
 .output
  .vptr_info
   .path_name: stack_1   .delimiter: |

 ptr_value =  231|0   000231100043 000000000000

  .ptr_value_info
   .segN:        231   .wordN: 0  .bitN: 0  .ringN: 1
   .inner_ring:          T   .needs_terminate:  T
   .readable      at  231|0  for 4096 (10000_o) words

  .seg_info
   .dir_name:    >process_dir_dir>!BPbDxmKBBBBBBB
   .ent_name:    stack_1
   .ent_type:    seg
   .uid:                 602752511505
   .max_length:          4096 words
   .pages_used   4               .records_used:  4
   .raw_mode:    rew            .ring_brackets:  1,1,1
   .effective_mode:   null
   .dtcm:        05/09/26 17:51          .dtd:  12/31/00 17:00
   .dtu:         05/09/26 17:51         .dtem:  05/09/26 17:48


000000
stack_header           @ 432|0
  cpm_data_ptr = 0(0) [invalid],
  combined_stat_ptr = 326|0   [pd]>!BBBKbWmjwJHhMM.area.linker,
  clr_ptr = 326|0     [pd]>!BBBKbWmjwJHhMM.area.linker, max_lot_size = 1024,
  main_proc_invoked = 0, run_unit_depth = 0, cur_lot_size = 512,
  cpm_enabled = "000000"b3,
  system_free_ptr = 326|0   [pd]>!BBBKbWmjwJHhMM.area.linker,
  user_free_ptr = 326|0   [pd]>!BBBKbWmjwJHhMM.area.linker, null_ptr = null,
  stack_begin_ptr = 231|2000   [pd]>stack_1,
  stack_end_ptr = 231|2000[1]   [pd]>stack_1, lot_ptr = 231|0   [pd]>stack_1,
  signal_ptr = 256|14710   signal_$signal_, bar_mode_sp = 0(0) [invalid],
  pl1_operators_ptr = 260|21766   pl1_operators_$operator_table,
  call_op_ptr = 260|30267   pl1_operators_$alm_operators_begin,
  push_op_ptr = 260|30273   pl1_operators_$alm_push,
  return_op_ptr = 260|30317   pl1_operators_$alm_return,
  return_no_pop_op_ptr = 260|30325   pl1_operators_$alm_return_no_pop,
  entry_op_ptr = 260|30307   pl1_operators_$alm_entry,
  trans_op_tv_ptr = 326|5312[1]   [pd]>!BBBKbWmjwJHhMM.area.linker,
  isot_ptr = 231|1000   [pd]>stack_1, sct_ptr = 231|1000   [pd]>stack_1,
  unwinder_ptr = 256|11730   unwinder_$unwinder_, sys_link_info_ptr = null,
  rnt_ptr = 326|102[1]   [pd]>!BBBKbWmjwJHhMM.area.linker, ect_ptr = null,
  assign_linkage_ptr = 326|0   [pd]>!BBBKbWmjwJHhMM.area.linker,
  heap_header_ptr = null
  trace                @ 432|72
    frames             @ 432|72
      count = 0, top_ptr = 0|0 [invalid]
    in_trace = "000000000000"b3
  OFF: have_static_vlas
```

```
ring_zero_dump: pertinent elements from returned structure:

cv_ptr_details
 .output
  .vptr_info
   .path_name: stack_4   .delimiter: |

 ptr_value =  234|0   000234400043 000000000000

  .ptr_value_info
   .segN:        234   .wordN: 0  .bitN: 0  .ringN: 4
   .needs_terminate:  T
   .readable      at  234|0  for 23552 (56000_o) words
   .writeable     at  234|0  for 65536 (200000_o) words

  .seg_info
   .dir_name:    >process_dir_dir>!BPbDxmKBBBBBBB
   .ent_name:    stack_4
   .ent_type:    seg
   .uid:                 602752511515
   .max_length:          65536 words
   .pages_used   23              .records_used:  23
   .raw_mode:    rew            .ring_brackets:  4,4,4
   .effective_mode:   rew
   .dtcm:        05/09/26 17:51          .dtd:  12/31/00 17:00
   .dtu:         05/09/26 17:51         .dtem:  05/09/26 17:48


000000
stack_header           @ 234|0
  cpm_data_ptr = 0(0) [invalid],
  combined_stat_ptr = 247|0   [pd]>!BBBKbWmjwHnNWX.area.linker,
  clr_ptr = 247|0     [pd]>!BBBKbWmjwHnNWX.area.linker, max_lot_size = 1024,
  main_proc_invoked = 0, run_unit_depth = 0, cur_lot_size = 512,
  cpm_enabled = "000000"b3,
  system_free_ptr = 247|0   [pd]>!BBBKbWmjwHnNWX.area.linker,
  user_free_ptr = 247|0   [pd]>!BBBKbWmjwHnNWX.area.linker, null_ptr = null,
  stack_begin_ptr = 234|2000   [pd]>stack_4,
  stack_end_ptr = 234|30200   [pd]>stack_4, lot_ptr = 234|0   [pd]>stack_4,
  signal_ptr = 256|14710   signal_$signal_, bar_mode_sp = 0(0) [invalid],
  pl1_operators_ptr = 260|21766   pl1_operators_$operator_table,
  call_op_ptr = 260|30267   pl1_operators_$alm_operators_begin,
  push_op_ptr = 260|30273   pl1_operators_$alm_push,
  return_op_ptr = 260|30317   pl1_operators_$alm_return,
  return_no_pop_op_ptr = 260|30325   pl1_operators_$alm_return_no_pop,
  entry_op_ptr = 260|30307   pl1_operators_$alm_entry,
  trans_op_tv_ptr = 247|5312   !BBBKbWmjwHnNWX.area.linker$operator_pointers_,
  isot_ptr = 234|1000   [pd]>stack_4, sct_ptr = 234|1000   [pd]>stack_4,
  unwinder_ptr = 256|11730   unwinder_$unwinder_,
  sys_link_info_ptr = 247|22132   [pd]>!BBBKbWmjwHnNWX.area.linker,
  rnt_ptr = 247|102   [pd]>!BBBKbWmjwHnNWX.area.linker,
  ect_ptr = 247|17240   [pd]>!BBBKbWmjwHnNWX.area.linker,
  assign_linkage_ptr = 247|0   [pd]>!BBBKbWmjwHnNWX.area.linker,
  heap_header_ptr = null
  trace               @ 234|72
    frames            @ 234|72
      count = 0, top_ptr = 0|0 [invalid]
    in_trace = "000000000000"b3
  OFF: have_static_vlas
```

The main testing needed for dump_segment (which also covers ring_zero_dump) is to verify that the data returned in the cv_ptr_details.output sub-structure is correct; and that each pertinent element has been mapped properly onto the limit variables already used by dump_segment in determining which words it can display. Those tests will be done manually. The mapping will be visible in the compare_ascii comparison of old and new versions of dump_segment.pl1 and will be reviewed by the code auditor for correctness.

Code for dump_segment -control_args was reordered to separate arguments by type of control being selected. The new control argument categories are reflected in the revised dump_segment.info and ring_zero_dump.info. They are also summarized in the brief listing arguments displayed by giving one of these commands without any arguments. This summary is shown below.

```
dump_segment

Syntax as a command:
    ds VIRTUAL_POINTER {offset} {length} {-control_args}
    ds -name PATH      {offset} {length} {-control_args}

Syntax as an active function:
   [ds VIRTUAL_POINTER {offset} {length} {-control_args}]
   [ds -name PATH      {offset} {length} {-control_args}]

Arguments (data selection):
 VIRTUAL_POINTER   offset   length

Control arguments (data selection):
 -name PATH, -nm PATH   -entry_point NAME, -ep NAME   -block N, -bk N

Control arguments (data display as a command):
 -address, -addr        -offset {N}, -ofs {N}   -long, -lg
 -no_address, -nad      -no_offset, -nofs       -short, -sh
 -header, -he                                   -block N, -bk N
 -no_header, -nhe

Control arguments (data format):
 -interpreted, -it     -suppress_duplicates, -sd
 -no_interpret, -nit   -no_suppress_duplicates, -nsd
 -raw
 -no_raw, -nraw

Control arguments (structure display as a command):
 -as STRUCTURE_NAME   -in VIRTUAL_ENTRY   -long, -lg   -short, -sh

Control arguments (interpreted data):
 -4bit   -bcd   -character, -ch, -ascii   -ebcdic8     -ebcdic9

Control arguments (raw data):
 -hex8   -hex9   -octal, -oc

Notes:  For details, type:  help ds  or  help virtual_pointers.gi
```

Finally, dump_segment.pl1 was changed to create an ssu_ standalone invocation so ssu_ can emit its diagnostic messages and managing its use of temporary segments, areas, etc. This also allows cv_ptr_$details to share use of dump_segment's standalone invocation, and to diagnose problems found in the VIRTUAL_PTR argument with messages emitted on behalf of dump_segment or ring_zero_dump.

Those changes were reviewed for accuracy during development and will be audited as well. The control variables used by dump_segment.pl1 were not modified by these changes. Therefore, code throughout the program was mostly unchanged and need not be tested.

Testing bit_string.pl1

The bit_string (bit) command/AF was tested manually after a white-box inspection of the program. Conversion of a binary character string to a binary bit string is performed by PL/1 conversion software. So, testing of that conversion isn't necessary. Tests on need to prove that 0 and 1 digits are accepted, and other digits are rejected.

The bit_string command converts character strings in quaternary, octal, and hexadecimal radices by lookup of each input character in a table of accepted digits for the given input radix.

Tests to convert digits of each supported INPUT_STR radix verify that all supported digits of the four supported radices work correctly, and that a character not in the supported digits is diagnosed as an error.

The full set of tests are shown below. All tests produced expected results.

```
ec test_bit_string

Test acceptable -in radix values.

 OK      [ bit 1 -in b1                  ]  =  "1"b              Expected:  "1"b
 OK      [ bit 1 -in _b1                 ]  =  "1"b              Expected:  "1"b
 OK      [ bit 1 -in b                   ]  =  "1"b              Expected:  "1"b
 OK      [ bit 1 -in _b                  ]  =  "1"b              Expected:  "1"b

 OK      [ bit 3 -in b2                  ]  =  "11"b             Expected:  "11"b
 OK      [ bit 3 -in _b2                 ]  =  "11"b             Expected:  "11"b
 OK      [ bit 3 -in q                   ]  =  "11"b             Expected:  "11"b
 OK      [ bit 3 -in _q                  ]  =  "11"b             Expected:  "11"b

 OK      [ bit 7 -in b3                  ]  =  "111"b            Expected:  "111"b
 OK      [ bit 7 -in _b3                 ]  =  "111"b            Expected:  "111"b
 OK      [ bit 7 -in o                   ]  =  "111"b            Expected:  "111"b
 OK      [ bit 7 -in _o                  ]  =  "111"b            Expected:  "111"b

 OK      [ bit F -in b4                  ]  =  "1111"b           Expected:  "1111"b
 OK      [ bit F -in _b4                 ]  =  "1111"b           Expected:  "1111"b
 OK      [ bit F -in x                   ]  =  "1111"b           Expected:  "1111"b
 OK      [ bit F -in _x                  ]  =  "1111"b           Expected:  "1111"b

 OK      [ bit F -in b5                  ]  =  ERR               Expected:  ERR
 OK      [ bit F -in _b5                 ]  =  ERR               Expected:  ERR
 OK      [ bit F -in y                   ]  =  ERR               Expected:  ERR
 OK      [ bit F -in _y                  ]  =  ERR               Expected:  ERR


Test acceptable radix_indicator substrings to override -in

 OK      [ bit "1"b1 -in q               ]  =  "1"b              Expected:  "1"b
 OK      [ bit 1_b1 -in q                ]  =  "1"b              Expected:  "1"b
 OK      [ bit "1"b -in q                ]  =  "1"b              Expected:  "1"b
 OK      [ bit 1_b -in q                 ]  =  "1"b              Expected:  "1"b

 OK      [ bit "3"b2 -in o               ]  =  "11"b             Expected:  "11"b
 OK      [ bit 3_b2 -in o                ]  =  "11"b             Expected:  "11"b
 OK      [ bit 3q -in o                  ]  =  "11"b             Expected:  "11"b
 OK      [ bit 3_q -in o                 ]  =  "11"b             Expected:  "11"b

 OK      [ bit "7"b3 -in x               ]  =  "111"b            Expected:  "111"b
 OK      [ bit 7_b3 -in x                ]  =  "111"b            Expected:  "111"b
 OK      [ bit 7o -in x                  ]  =  "111"b            Expected:  "111"b
 OK      [ bit 7_o -in x                 ]  =  "111"b            Expected:  "111"b
```

```
OK      [ bit "F"_b4 -in b                ]  =  "1111"b              Expected:  "1111"b
OK      [ bit Fx -in b                    ]  =  "1111"b              Expected:  "1111"b
OK      [ bit F_x -in b                   ]  =  "1111"b              Expected:  "1111"b

OK      [ bit "F"_b5 -in b                ]  =  ERR                  Expected:  ERR
OK      [ bit Fy -in b                    ]  =  ERR                  Expected:  ERR
OK      [ bit F_y -in b                   ]  =  ERR                  Expected:  ERR


Test acceptable input digits for each radix.

OK      [ bit 0 -in b                     ]  =  "0"b                 Expected:  "0"b
OK      [ bit 1 -in b                     ]  =  "1"b                 Expected:  "1"b
OK      [ bit 2 -in b                     ]  =  ERR                  Expected:  ERR

OK      [ bit 0 -in q                     ]  =  "00"b                Expected:  "00"b
OK      [ bit 1 -in q                     ]  =  "01"b                Expected:  "01"b
OK      [ bit 2 -in q                     ]  =  "10"b                Expected:  "10"b
OK      [ bit 3 -in q                     ]  =  "11"b                Expected:  "11"b
OK      [ bit 4 -in q                     ]  =  ERR                  Expected:  ERR

OK      [ bit 0 -in o                     ]  =  "000"b               Expected:  "000"b
OK      [ bit 1 -in o                     ]  =  "001"b               Expected:  "001"b
OK      [ bit 2 -in o                     ]  =  "010"b               Expected:  "010"b
OK      [ bit 3 -in o                     ]  =  "011"b               Expected:  "011"b
OK      [ bit 4 -in o                     ]  =  "100"b               Expected:  "100"b
OK      [ bit 5 -in o                     ]  =  "101"b               Expected:  "101"b
OK      [ bit 6 -in o                     ]  =  "110"b               Expected:  "110"b
OK      [ bit 7 -in o                     ]  =  "111"b               Expected:  "111"b
OK      [ bit 8 -in o                     ]  =  ERR                  Expected:  ERR

OK      [ bit 0 -in x                     ]  =  "0000"b              Expected:  "0000"b
OK      [ bit 1 -in x                     ]  =  "0001"b              Expected:  "0001"b
OK      [ bit 2 -in x                     ]  =  "0010"b              Expected:  "0010"b
OK      [ bit 3 -in x                     ]  =  "0011"b              Expected:  "0011"b
OK      [ bit 4 -in x                     ]  =  "0100"b              Expected:  "0100"b
OK      [ bit 5 -in x                     ]  =  "0101"b              Expected:  "0101"b
OK      [ bit 6 -in x                     ]  =  "0110"b              Expected:  "0110"b
OK      [ bit 7 -in x                     ]  =  "0111"b              Expected:  "0111"b
OK      [ bit 8 -in x                     ]  =  "1000"b              Expected:  "1000"b
OK      [ bit 9 -in x                     ]  =  "1001"b              Expected:  "1001"b
OK      [ bit a -in x                     ]  =  "1010"b              Expected:  "1010"b
OK      [ bit b -in x                     ]  =  "1011"b              Expected:  "1011"b
OK      [ bit c -in x                     ]  =  "1100"b              Expected:  "1100"b
OK      [ bit d -in x                     ]  =  "1101"b              Expected:  "1101"b
OK      [ bit e -in x                     ]  =  "1110"b              Expected:  "1110"b
OK      [ bit f -in x                     ]  =  "1111"b              Expected:  "1111"b
OK      [ bit A -in x                     ]  =  "1010"b              Expected:  "1010"b
OK      [ bit B -in x                     ]  =  "1011"b              Expected:  "1011"b
OK      [ bit C -in x                     ]  =  "1100"b              Expected:  "1100"b
OK      [ bit D -in x                     ]  =  "1101"b              Expected:  "1101"b
OK      [ bit E -in x                     ]  =  "1110"b              Expected:  "1110"b
OK      [ bit F -in x                     ]  =  "1111"b              Expected:  "1111"b
OK      [ bit g -in x                     ]  =  ERR                  Expected:  ERR
OK      [ bit G -in x                     ]  =  ERR                  Expected:  ERR


Test setting length during radix conversion.
These also test what happens if a stringsize on-unit returns to let bit_string handle the condition.

OK      [ bit 051                         ]  =  "000101001"b         Expected:  "000101001"b
OK      [ bit 051 -ln 12                  ]  =  "000101001000"b      Expected:  "000101001000"b
OK      [ bit 051 -out x                  ]  =  "148"b4              Expected:  "148"b4   stringsize signalled
OK      [ bit 051 -ln 12 -out x           ]  =  "148"b4              Expected:  "148"b4
OK      [ bit 051 -ln 8 -out x            ]  =  "14"b4               Expected:  "14"b4
OK      [ bit 051 -ln 9 -out x            ]  =  "148"b4              Expected:  "148"b4   stringsize signalled
```

```
Test use of -quote (-q), -no_quote (-nq) and -radix (output format controls)

 OK      [ bit 051 -ln 12 -q              ]  =  "000101001000"b       Expected:  "000101001000"b
 OK      [ bit 051 -ln 12 -nq -out q      ]  =  011020               Expected:  011020
 OK      [ bit 051 -ln 12 -radix -out q   ]  =  011020_q             Expected:  011020_q


Test handling of unknown -control_arg

 OK      [ bit 051 -ln 12 -foobar -radix  ]  =  ERR                  Expected:  ERR
           The bit_string command/af emits the following error message...
           bit_string: Specified control argument is not accepted.  -foobar
```

Documentation

For the new bit_string (bit) command/active-function, the following documentation is proposed as an info segment.

```
2026-05-10  bit_string, bit

Syntax as a command:
   bit INPUT_STR  {-control_args}

Syntax as an active function:
  [bit INPUT_STR  {-control_args}]

Function:
This command or active function accepts a character representation of a
pl1 bit string as an input value.

The program converts INPUT_STR to an actual bit string using either its
ending radix indicator or the -in control argument to specify the radix
in which INPUT_STR characters are expressed.

By default, the result bit string is displayed or returned as a pl1
quoted bit string with a binary radix factor:  "10011"b

Arguments:
INPUT_STR
   is the character representation of the input bit string.  It may end
   with one of the radix indicators in the "List of radix_indicator
   substrings" section below.

   If INPUT_STR does not end with a radix indicator and the -in control
   argument is not given, the INPUT_STR is assumed to be expressed in
   octal characters.

Control arguments (radix):
-in RADIX_INDICATOR
   controls how characters of INPUT_STR are interpreted, unless that
   string ends with its own radix_indicator substring.  An indicator
   can specify binary, quaternary, octal or hexadecimal characters in
   the INPUT_STR.  See the "List of radix_indicator substrings" section
   for details.  (default: octal input)
-out RADIX_INDICATOR
   displays or returns the result bit string as a character string
   using characters of one of the power-of-2 bases.  See the
   "List of radix_indicator substrings" section for details.
   (default: binary output)
```

```
Control arguments (output format):
-length L, -ln L
   gives a desired bit length for the resulting bit string.  By
   default, each INPUT_STR character is converted to B-bits according
   to its designated input radix:

      Radix          B
      -----------  --------------------
      binary       1 bit  per character
      quaternary   2 bits per character
      octal        3 bits per character
      hexadecimal  4 bits per character

   Result default length D = B * length(INPUT_STR).  If INPUT_STR ends
   with its own radix_indicator substring, that substring is excluded
   from the length(INPUT_STR) determination.

   If L is greater than the default length D, then L-D zero bits are
   appended to the right end of the result.  If L is less than the
   default length D, then D-L bits are removed from the right end of
   the result.

-quote, -q
   display or return the result as a pl1 quoted bit string with
   trailing radix factor. (default)  For example:
       "1011"b  or  "23"b2  or  "54"b3  or  "b"b4
-no_quote, -nq
   display or return the result as a character string without
   surrounding quote characters or pl1 radix factor:
       1011  or  23  or  54  or  b
-radix
   display or return the result as a character string without
   surrounding quote characters but with a radix indicator:
      1011_b  or  23_q  or  54_o  or  b_x

List of radix_indicator substrings:
   INPUT_STR may end with one of the following strings which indicate
   which numbering system's digits are expressing the value.
   Indicator characters may also be given in uppercase.  -in and -out
   may be followed by any of the values below; but values beginning
   with underscore (_) are not usually given with these -control_args.

b1, b, _b
   the number is interpreted as a base two string (binary).
b2, q, _q
   the number is interpreted as a base four string (quaternary).
b3, o, _o
   the number is interpreted as a base eight string (octal).
```

```
b4, x, _x
   the number is interpreted as a base sixteen string (hexadecimal).

   The b4 format may be appended to an INPUT_STR only if the entire
   string is surrounded by double-quote (") characters.  Without this
   extra level of quoting, the b4 radix factor would be treated as
   digits in the INPUT_STR string.

   In the example   bit_string "a1"b4  the command processor removes
   the quotes before invoking bit_string.  The bit_string program
   receives  a1b4  which it treats as a  4-character octal number.  A
   conversion error occurs.

   An extra layer of quotes must surround the INPUT_STR:
      bit_string """a1""b4"
   causing the command to receive:  "a1"b4
   It may be easier to enter a hexadecimal string as a numeric string
   followed by the x radix indicator:        bit_string a1x
   or as a numeric string with -in control:  bit_string a1 -in b4

Notes:
To express a result bit string in a non-binary radix, each output digit
represents a group of B-bits in the result (where B is 2, 3, or 4).  If
the length of the result is not evenly divisible by B, then a
stringsize condition is signalled.  If the on-unit handing this
condition returns to the point of error, then length of the result is
extended by adding B - mod(L, B) zeros to the right end of the result
to enable correct display of the last digit in the result.  Note that
the system default handler for stringsize silently returns to the point
of error, thus enabling a correct result to be displayed.

Examples:
To convert an octal output word from the ring_zero_dump command as a
binary bit string:

string [ring_zero_dump 71|3357 0 1]
400000000000

bit_string [ring_zero_dump 71|3357 0 1] -in o
"100000000000000000000000000000000000"b

This same result could be displayed for this input if a trailing
radix indicator string were given at the end of the first argument.

bit [ring_zero_dump 71|3357 0 1]o
"100000000000000000000000000000000000"b

Since the default INPUT_RADIX is octal and the default OUTPUT_RADIX is
binary, only the INPUT_STR need be given.

bit [ring_zero_dump 71|3357 0 1]
"100000000000000000000000000000000000"b

To express an octal bit string properly in hexadecimal, "0"b bits must
```

```
be added to the input bit string to fill out the final hex digit.  This
is shown in the example below.

bit 051o
"000101001"b

bit 051o -length 12
"000101001000"b

bit 051o -length 12 -out b4
"148"b4

Extending the length of the result bit string would occur with the
default handling for the stringsize condition, as shown below.

bit 051 -out b4
"148"b4

To enter INPUT_STR values in hexadecimal using the b4 radix indicator,
the hex digits of INPUT_STR must be surrounded by double-quote (")
characters to separate the hex digits from the b4 radix indicator.
Otherwise, 1ab4 would be interpreted as an octal INPUT_STR with invalid
digits (digits that were not 0, 1, 2, 3, 4, 5, 6, or 7).

bit_string 1ab4
bit_string: Error in conversion.  Invalid digits for base = 8.
 INPUT_STR:  1ab4
r 09:17 0.035 0

bit_string "1a"b4
bit_string: Error in conversion.  Invalid digits for base = 8.
 INPUT_STR:  1ab4
r 09:18 0.034 0

bit_string """1a""b4"
"00011010"b
r 09:26 0.026 0

It is usually easier to enter a hexadecimal bit string as a hex number
using the x or _x radix indicator.

bit_string 1a_x
"00011010"b
r 09:30 0.015 0

bit_string 1ax
"00011010"b
r 09:30 0.013 0
```

For the change cv_ptr_ function, with its new cv_ptr_$ringno and cv_ptr_$details entry points, here is the entire proposed info segment.

```
2026-04-17  cv_ptr_

The cv_ptr_ function converts a virtual pointer to a pointer value.  A
virtual pointer is a character-string representation of a pointer
value.

See also the virtual_pointer (vptr) command/active-function.  When used
in a Multics command line, it converts a virtual pointer to a
segno|W(B) virtual pointer of a segment initiated in the user process.

Entry points in cv_ptr_:
 2026-04-17  cv_ptr_           2025-10-10  cv_ptr_$terminate
 2025-10-11  cv_ptr_$ringno    2026-04-17  cv_ptr_$details


Entry:  cv_ptr_   (64 lines in entry point; 3 other entry points in info)

2026-04-17  cv_ptr_

Function:
A virtual pointer is a character-string representation of a pointer
value.  The cv_ptr_ function converts its vptr argument (a virtual
pointer) to a real pointer value.
 - The vptr string is divided into segment_identifier and offset
   substrings.
 - The segment identified by vptr is initiated with a null reference
   name.
 - Then a ptr_value pointing to the specified offset within that
   segment is returned to the caller.

The ptr_value returned by the cv_ptr_ function cannot be used to call a
program entry point.  The cv_ptr_ function constructs the returned
pointer to a segment in a way that avoids copying of the segment's
linkage and internal static data into the combined linkage area of the
caller's process.  Use the cv_entry_ function to convert a virtual
entry character string to a callable entry value.

When the calling program no longer needs to reference the segment, it
should call the cv_ptr_$terminate subroutine to remove the segment's
null reference name in the ring in which the segment was made known
to the calling process.

Syntax:
declare cv_ptr_ entry (char(*), fixed bin(35)) returns (ptr);

ptr_value = cv_ptr_ (vptr, code);
```

```
Arguments:
vptr
   is the virtual pointer to be converted.  (Input)  The pointer may
   contain one or two components:  a segment identifier; and an
   optional offset into the segment.  For information on virtual
   pointer formats, type:  help virtual_pointers.gi
code
   is a standard status code.  (Output)
ptr_value
   is an aligned pointer to the segment identified by the vptr string.
   (Output)

Notes on the returned ptr_value:
To examine the parts of the returned ptr_value, use the PL/I
segno, wordno or bitno builtin functions.  For details on these
non-standard PL/I builtins, type:  help pl1_new_features.

To examine the effective ring number stored in the ptr_value, use the
cv_ptr_$ringno function.

The ptr_value returned by the cv_ptr_ function will have an effective
ring number R >= current ring of execution (the number of the
protection ring in which the caller of cv_ptr_ is executing).  When
trying to reference an inner-ring segment, the Multics access control
hardware sets the effective ring in the returned ptr_value to the
caller's ring of execution.  This protects the inner-ring segment if
the caller tries to reference that segment directly using the returned
pointer.  However, any null ptr_value returned by cv_ptr_$details has
its ring number set to 0.  This ensures that all null pointer values
have the same content irrespective of which ring sets the pointer to
null.

Contents of an inner-ring segment may only be referenced by using the
ring_zero_peek_ subroutine, or the ring_zero_dump (rzd) command.


Entry:  cv_ptr_$ringno   (18 lines in entry point; 3 other entry points in
info)

2025-10-11  cv_ptr_$ringno

Function:
The cv_ptr_$ringno function returns the effective ring number R from
an input pointer P.

Syntax:
declare cv_ptr_$ringno entry (ptr) returns (fixed bin(3, 0));

R = cv_ptr_$ringno (P);

Arguments:
P
   is a scalar aligned pointer value.
R
```

is a fixed binary (3, 0) value holding the effective ring number value from the input pointer P. This will be a value in the range 0 through 7.

Entry: cv_ptr_$terminate (16 lines in entry point; 3 other entry points in info)

2025-10-10 cv_ptr_$terminate

Function: This entry point is called to terminate the segment that has been initiated by a previous call to cv_ptr_. It will not attempt to terminate a null pointer or a ring-0 segment. It will silently fail to terminate an inner-ring segment.

Syntax:
```
declare cv_ptr_$terminate (ptr);

call cv_ptr_$terminate (ptr_value);
```

Arguments:
ptr_value
is the scalar aligned pointer returned by the previous call to cv_ptr_. (Input)

Entry: cv_ptr_$details (405 lines in entry point; 3 other entry points in info)

2026-04-17 cv_ptr_$details

Function:
A virtual pointer is a character-string representation of a pointer value. The cv_ptr_$details function converts its vptr argument (a virtual pointer) to a real pointer value.
- **The vptr string is divided into segment_identifier and offset substrings.**
- **The segment identified by vptr is initiated with a null reference name if not already known to the process.**
- **Then a ptr_value pointing to the specified offset within that segment is returned to the caller.**

The cv_ptr_$details entry point displays an error message if a problem is found in the virtual pointer string; or it returns the ptr_value with additional information about the segment referenced by ptr_value.

A ptr_value returned by the cv_ptr_$details function cannot be used to call a program entry point. The function constructs the returned pointer to a segment in a way that avoids copying of the segment's linkage and internal static data into the combined linkage area of the caller's process. Use the cv_entry_ function to convert a virtual entry character string to a callable entry value.

When the calling program no longer needs to reference the segment, it

```
should terminate that segment if cv_ptr_details.needs_terminate = "1"b.
The caller should use the cv_ptr_$terminate subroutine to remove the
segment's null reference name in the ring in which the segment was made
known to the calling process.

Syntax:
declare cv_ptr_$details entry (char(*), ptr, fixed bin(35))
   returns(ptr);

ptr_value = cv_ptr_$details (vptr, details_ptr, code);

Arguments:
vptr
   is the virtual pointer to be converted. (Input)  The pointer may
   contain one or two components:  a segment identifier; and an
   optional offset into the segment.  For information on virtual
   pointer formats, type:  help virtual_pointers.gi

details_ptr
   is a pointer to a cv_ptr_details structure in which the caller
   provides additional inputs, and receives data about the ptr_value
   segment. (Input)  See the section titled "Notes on calling
   cv_ptr_$details" section for more information.
code
   is a standard status code which is non-zero if an error was detected
   while converting vptr to a pointer value.  (Output)  For information
   about these errors, see the "List of error codes" section.
ptr_value
   is an aligned pointer to the segment identified by the vptr string.
   (Output)  This segment was: initiated in the caller's ring of
   execution; or was found or initiated in an inner ring.

List of error codes:
   The following error codes must be handled by the caller.  They
   report internal errors in caller's code, or conditions whose impact
   depends upon the caller's intended use of the ptr_value returned by
   cv_ptr_$details.  Other codes may also report errors found while
   parsing the vptr input.  cv_ptr_ reported these errors to the user.
   The caller should just abort the current operation.
error_table_$null_info_ptr
   cv_ptr_$details was called with details_ptr = null().
error_table_$unimplemented_version
   cv_ptr_$details was called with cv_ptr_details.version set to an
   unknown value.

error_table_$bad_command_name
   cv_ptr_details.sci_ptr is invalid and cv_ptr_details.command_name
   has not been set.
```

```
error_table_$boundviol
   The vptr string references a location which is not within the
   storage limits of the identified segment.  The caller may need to
   extend those limits in order to reference that location.

Notes on calling cv_ptr_$details:
cv_ptr_$details uses the facilities of an ssu_ invocation to:
 - provide an area segment which will temporarily hold descriptors for
   the various parts of the vptr argument; and
 - display diagnostic messages on behalf of the caller if the vptr
   conversion fails.

If the caller passes a non-null sci_ptr argument, cv_ptr_ will obtain
the area from that subsystem; and will release that area before
returning to the caller.

The details_ptr input argument points to a cv_ptr_details structure
that includes input and output sub-structures.  When the caller passes
a null sci_ptr argument, elements in the cv_ptr_details.input
substructure provide information about the caller needed to create an
ssu_ standalone invocation.  cv_ptr_ uses that invocation to:
 - get its allocation area;
 - report vptr failure messages on behalf of that caller.
It then destroys the invocation before returning to the caller.

A caller preferring no error messages describing a conversion failure
should use the cv_ptr_ entry point rather than cv_ptr_$details.  That
entry point just returns a Multics status code if conversion of the
virtual pointer string fails.

Notes on the returned ptr_value:
To examine the parts of the returned ptr_value, use the PL/I
segno, wordno or bitno builtin functions.  For details on these
non-standard PL/I builtins, type:  help pl1_new_features.

To examine the effective ring number stored in the ptr_value, use the
cv_ptr_$ringno function.

The ptr_value returned by the cv_ptr_$details function will have an
effective ring number R >= current ring of execution (the number of the
protection ring in which the caller of cv_ptr_ is executing).  When
trying to reference an inner-ring segment, the Multics access control
hardware sets the effective ring in the returned ptr_value to the
caller's ring of execution.  This protects the inner-ring segment if
the caller tries to reference that segment directly using the returned
pointer.  However, any null ptr_value returned by cv_ptr_$details has
its ring number set to 0.  This ensures that all null pointer values
have the same content irrespective of which ring sets the pointer to
null.

Contents of an inner-ring segment may only be referenced by using the
ring_zero_peek_ subroutine, or the ring_zero_dump (rzd) command.

Notes on details_ptr:
```

```
If the details_ptr argument is non-null, it should point to the
structure shown below.  The cv_ptr_details.incl.pl1 include segment
provides this declaration.  See the sections beginning with
"List of cv_ptr_details" for information about these elements.

  dcl 1 cv_ptr_details        aligned,
      2 input,
        3 version             char(16),
        3 sci_ptr             ptr,
        3 arg_list_ptr        ptr,
        3 program_name        char(32) var,
        3 display_output      bit(1) aligned,
        3 pad0                (2) fixed bin,

      2 output,
        3 vptr_info,
          4 path_name         char(202) var,
          4 ref_name          char(32) var,
          4 seg_number        char(6) var,
          4 delimiter         char(1) var,
          4 ep_name           char(259) var,
          4 offset_numbers    char(13) var,
          4 pad1              bit(1) aligned,

        3 ptr_value_info,
          4 segN              fixed bin(12),
          4 wordN             fixed bin(18),
          4 bitN              fixed bin(6),
          4 ringN             fixed bin(3),
          4 null_value        bit(1) aligned,
          4 needs_terminate   bit(1) aligned,
          4 hardcore          bit(1) aligned,
          4 inner_ring        bit(1) aligned,

          4 word_aligned      bit(1) aligned,
          4 readable,
            5 read_before     fixed bin(35),
            5 read_after      fixed bin(35),
          4 writeable,
            5 write_before    fixed bin(35),
            5 write_after     fixed bin(35),
          4 pad2              (2) bit(1) aligned,

        3 seg_info,
          4 ent_type          fixed bin(2),
          4 dir_name          char(168) unaligned,
          4 ent_name          char(32) unaligned,
          4 arch_comp_name    char(32) unaligned,
          4 arch_comp_offset  fixed bin(18),
          4 msf_selected_comp fixed bin(24),
          4 msf_total_comps   fixed bin(24),
          4 bit_count         fixed bin(24),
          4 max_length        fixed bin(35),
          4 raw_mode          bit(5) aligned,
```

```
          4 effective_mode    bit(5) aligned,
          4 rb                (0:2) fixed bin(6),

          4 lvid              bit(36),
          4 uid               bit(36),
          4 dtcm              bit(36),
          4 dtu               bit(36),
          4 dtd               bit(36),
          4 dtem              bit(36),
          4 pages_used        fixed bin(12),
          4 records_used      fixed bin(18),
          4 pad3              (5) fixed bin(35);

  dcl  Details_Version_1      char(16) internal static
                              options(constant)
                              init("cv_ptr_details_1");
```

List of cv_ptr_details.input elements:

The four elements below are input values supplied by the caller of cv_ptr_$details.

version

gives the version of this structure. It should be set to the Details_Version_1 scalar value. (Input)

sci_ptr

is an optional pointer to an ssu_ subsystem control structure. (Input) Callers with their own ssu_ invocation should assign that subsystem's sci_ptr to this element. Callers not part of an ssu_ subsystem set this element to a null pointer, and set the next two elements. This information is used to create the ssu_ subsystem invocation needed by cv_ptr_$details to parse the vptr string and report any errors to the user.

arg_list_ptr

points to the calling program's argument list. This value tells the ssu_ invocation whether to generate error messages that come from: a command or subroutine; or from an active function. (Input) This element is ignored if the sci_ptr element is a non-null pointer. If arg_list_ptr is set to a null value, then ssu_ uses the stack_frame.arg_ptr element in the caller's stack frame as a pointer to the caller's argument list.

program_name

gives the name of the calling command, active function program, or subroutine. cv_ptr_ error messages are identified as coming from this program. (Input) This element is ignored if the sci_ptr element is a non-null pointer.

display_output

if "1"b, then cv_ptr_$details displays elements of the cv_ptr_details structure before returning to the calling program.

```
List of cv_ptr_details.vptr_info elements:
   These character strings hold the various parts found when parsing
   the virtual pointer string.  They are output values set by
   cv_ptr_$details just before it returns to the caller.  If the code
   argument is non-zero, these output values have no meaning.
path_name
   non-empty if the segment identifier has attributes of a relative or
   absolute pathname. (Output)
ref_name
   non-empty if the segment identifier has attributes of a reference
   name. (Output)
seg_number
   non-empty if the segment identifier has attributes of a segment
   number. (Output)

delimiter
   gives the delimiter character found between the segment identifier
   and offset parts of vptr.  It will be: a vertical bar (|); a dollar
   sign ($); or an empty string if there was no delimiter part found
   in vptr. (Output)
ep_name
   non-empty if the offset part of vptr has the attributes of an entry
   point name. (Output)

offset_numbers
   non-empty if the offset part of vptr has numeric values for word
   offset, bit offset within a word, and ring number. (Output)  These
   values are displayed in format:  W(B)[R]
   where:
    W  is an octal number of words from start of seg.  range[0:777777]
    B  is a decimal number of bits from start of word. range[0:35]
    R  is a decimal ring number of highest ring in
       which the pointer may be referenced.            range[0:7]

List of cv_ptr_details.ptr_value_info elements:
   These elements give attributes of the segment referenced in the
   ptr_value return argument.  If the code argument is non-zero,
   these output values have no meaning. (Output)
segN
   segment number of the selected segment.       range[-1:4095  ]
wordN
   word offset within selected segment.          range[ 0:262143]
bitN
   bit offset  within word of selected segment.  range[ 0:35    ]
ringN
   highest ring number in which ptr_value may
   be accessed.                                  range[ 0:7     ]

null_value
   vptr identifies a null pointer value. (Output)  A segment fault
   occurs if the caller references through a null ptr_value return
   argument.
```

```
needs_terminate
   segment referenced by ptr_value was initiated by cv_ptr_$details and
   should be terminated by calling cv_ptr_$terminate when the caller no
   longer needs the ptr_value return argument. (Output)

hardcore
   if "1"b, segment resides in ring-0 and is part of the Multics
   supervisor address space shared by all processes. (Output)  Contents
   of the segment may only be referenced by using the ring_zero_peek_
   subroutine.
inner_ring
   if "1"b, segment resides in a ring lower than that of the calling
   program but outside of ring-0. (Output)  Contents of such segment
   may only be referenced by using the ring_zero_peek_ subroutine.

word_aligned
   alignment of data referenced by vptr.  (Output)  If "1"b, then
   ptr_value points to word-aligned data and .readable and .writeable
   elements values are expressed in words.  If "0"b, then ptr_value
   points to non-word-aligned bits of data and .readable and .writeable
   element values are expressed in bits.
readable.read_before
   count of words or bits accessible before ptr_value. (Output)
readable.read_after
   count of words or bits accessible from ptr_value onward to the end
   of the referenced segment or archive component. (Output)

writeable.write_before
   count of words or bits writeable before ptr_value. (Output)
writeable.write_after
   count of words or bits writeable from ptr_value onward to the end
   of the referenced segment. (Output)  An archive component is not
   writeable by any program other than the archive command and its
   supporting utilities.

List of cv_ptr_details.seg_info elements:
   These values are attributes of the segment referenced by ptr_value
   as returned by hcs_$status_long.  If the code argument is non-zero,
   these output values have no meaning.
ent_type
   type of segment pointed to by the returned ptr_value. (Output)
     0: Link   1: Segment   2 Directory   3: MSF (multi-segment file)
   See the cv_ptrs_details.incl.pl1 include segment for declarations of
   these named constants, and the Entry_Type_Name array.
dir_name
   directory portion of segment's absolute path name. (Output)  This
   may be an empty string for a hardcore segment if it resides in
   memory rather than as a segment in the Multics file system.
ent_name
   entryname portion of segment's path name. (Output)
```

```
arch_comp_name
   component part of an archive path name. (Output)  Such names begin
   with entryname of an archive segment separated by :: from the name
   of component segment in that archive.  The name selects the
   comp_name component to be pointed to by the returned ptr_value.  The
   offset part of vptr refers to a location within that archive
   component.
arch_comp_offset
   offset in words of first word of the selected archive component
   within its containing archive segment. (Output)  This element is
   non-zero only if an archive pathname was found in vptr.

msf_selected_comp
   number of the MSF component segment actually selected when a
   multi-segment file (MSF) is found in vptr.  This is almost always
   component 0, the first component segment of an MSF.
msf_total_comps
   total number of component segments found in the selected
   multi-segment file (MSF).

bit_count
   current bit count set for the selected segment or archive component;
   or the MSF indicator count set for a multi-segment file; or 0 for a
   link or directory. (Output)
max_length
   max_length (in words) set on the selected segment or archive
   component.  The segment's length may not be extended beyond this
   value.

raw_mode
   "rew" or "sma" access bits from segment's ACL entry selected by the
   current process's access. (Output)  See the cv_ptrs_details.incl.pl1
   include segment for declarations of named constants for raw_mode bit
   values.
effective_mode
   effective access mode to the segment based upon caller's validation
   level (ring-of-execution) and AIM access_class of the process.
   (Output)  For an MSF entry, this is caller's effective access to
   component 0 of the multi-segment file.

rb
   array of ring brackets for the selected segment. (Output)
         0 <= ring-of-execution <= rb(0) gives write-bracket.
         0 <= ring-of-execution <= rb(1) gives read-bracket.
     rb(0) <= ring-of-execution <= rb(1) gives execute-bracket
     rb(1) <  ring-of-execution <= rb(2) gives gate-bracket.
         0 <= ring-of-execution <  rb(0) gives outward-call bracket.
   For a directory segment, only rb(0) and rb(1) are meaningful.
   For an MSF, the rbs are those on component 0 of the multi-segment
   file.
```

```
lvid
   logical volume ID for storage system volume on which the segment is
   stored. (Output)
uid
   unique ID for the entry. (Output)
dtcm
   date/time contents of segment were last modified. (Output)
dtu
   date/time contents of segment were last used. (Output)
dtd
   date/time contents of segment were last dumped to a file system
   backup tape. (Output)
dtem
   date/time at which the directory entry describing the segment was
   last modified. (Output)

pages_used
   highest page number of segment containing NONZERO data. (Output)
records_used
   count of NONZERO data pages found in the segment. (Output)
```

For the dump_segment (ds) command/active-function, the following changes are proposed in its info segment.

Similar changes are proposed for the ring_zero_dump (rzd) command/active-function, which has the same interface as dump_segment but uses ring_zero_peek_ to copy inner-ring data segment contents into the current ring of execution, so it can then be displayed. rzd.info changes will therefore not be shown in this MCR.

```
compare_ascii >doc>info>dump_segment.info ==

A1        11/26/87  dump_segment, ds
A2
A3        Syntax as a command:
A4                  ds segname {offset} {length} {-control_args}
A5                  ds segno {offset} {length} {-control_args}
A6                  ds virtual_pointer {offset} {length} {-control_args}
Changed by B to:
B1        :Info: dump_segment: ds:  2026-01-12  dump_segment, ds
B2
B3        Syntax as a command:
B4            ds VIRTUAL_POINTER {offset} {length} {-control_args}
B5            ds -name PATH      {offset} {length} {-control_args}


A10          [ds segname {offset} {length} {-control_args}]
A11          [ds segno {offset} {length} {-control_args}]
A12          [ds virtual_pointer {offset} {length} {-control_args}]
Changed by B to:
B9           [ds VIRTUAL_POINTER {offset} {length} {-control_args}]
B10          [ds -name PATH      {offset} {length} {-control_args}]


A17       optionally print out an edited version of the ASCII, BCD, EBCDIC (in
Changed by B to:
B15       optionally prints out an edited version of the ASCII, BCD, EBCDIC (in


A23       Arguments:
A24       segname
A25          is either a pathname or an octal segment number to the segment to
A26          be dumped. To specify a segment name that consists entirely of
A27          octal digits, the name must be preceded by the -name control
A28          argument.
A29       offset
A30          is the (octal) offset of the first word to be dumped.  If both
A31          offset and length are omitted, the entire segment is dumped.
A32       length
A33          is the (octal) number of words to be dumped.  If you supply offset
A34          and omit length, one word is dumped.
A35
A36
A37       segno
A38          is the octal segment number of a segment to be dumped. It cannot
A39          be a hardcore segment number.
A40       virtual_pointer
A41          is an ASCII representation of a pointer. See
A42          virtual_pointers.gi.info for further descriptions of virtual
A43          pointers.
A44
A45
```

```
A46       Control arguments:
A47       -block N, -bk N
A48          dumps words in blocks of N words separated by a blank line.  The
A49          offset, if being printed, is reset to the initial value at the
A50          beginning of each block.
Changed by B to:
B21       Arguments (data selection):
B22       VIRTUAL_POINTER
B23          is an ASCII representation of a pointer.  For a further description,
B24          type:  help virtual_pointers.gi
B25       offset
B26          is the octal offset of the first word to be dumped.  If both offset
B27          and length are omitted, the entire segment is dumped.
B28       length
B29          is the octal number of words to be dumped.  If you supply offset and
B30          omit length, one word is dumped.
B31
B32
B33       Control arguments (data selection):
B34       -name PATH, -nm PATH
B35          given in place of VIRTUAL_POINTER, PATH is treated as a segment
B36          pathname even though it may look like an octal segment number or a
B37          reference name.


A54          Use -entry_point only for object segments (created by a compiler or
A55          by the create_data_segment command).
A56       -name PATH, -nm PATH
A57          indicates that PATH is a pathname even though it may look like an
A58          octal segment number.
A59
A60
A61       Control arguments for display:
Changed by B to:
B41          If the offset argument is also given, it is added to the location
B42          selected by NAME.  Use -entry_point only for object segments
B43          (created by a compiler or by the create_data_segment command).
B44       -rest
B45          prints from any given offset (specified by the offset argument) to
B46          the end of the segment.
B47
B48
B49       Control arguments (data display as a command):
B50          The following control arguments may not be given when dump_segment
B51          is invoked as an active function.


Inserted in B:
B55       -no_address, -nad
B56          does not print the address.
B57
B58
Preceding:
A65       -header, -he


A71       -interpreted, -it
A72          prints the data decoded into the indicated format.
A73
A74
A75       -long, -lg
A76          prints eight words on a line. Four is the default. This control
A77          arguement cannot be used with -4bit, -bcd, -character, -ebcdic8,
```

```
A78          -ebcdic9, or -short. (Its use with these control arguments, other
A79          than -short, results in a line longer than 132 characters.)
A80       -no_address, -nad
A81          does not print the address.
Deleted by B, preceding:
B65       -no_header, -nhe


A85       -no_interpret, -nit
A86          suppresses printing of the decoded data.  (Default)
Changed by B to:
B68
B69
B70       -offset {N}, -ofs {N}
B71          prints the offset (relative to N words before the start of the data
B72          block being dumped) along with the data.  If you supply no N, 0 is
B73          assumed.


A91       -no_raw, -nraw
A92          suppresses printing of the raw data.
A93       -no_suppress_duplicates, -nsd
A94          indicates that sequential lines are to be printed even if they would
A95          be identical to previous lines.
A96       -offset {N}, -ofs {N}
A97          prints the offset (relative to N words before the start of the data
A98          block being dumped) along with the data.  If you supply no N, 0 is
A99          assumed.
A100      -raw
A101         indicates that the raw data is to be printed.  (Default)
A102      -rest
A103         prints from any given offset (specified by -offset or the offset
A104         argument) to the end of the segment.
A105
A106
Changed by B to:
B78       -long, -lg
B79          prints eight words on a line.  Four is the default.  This control
B80          arguement cannot be used with -4bit, -bcd, -character, -ebcdic8,
B81          -ebcdic9, or -short.  (Its use with these control arguments, other
B82          than -short, results in a line longer than 132 characters.)


Inserted in B:
B88       -block N, -bk N
B89          dumps words in blocks which are N octal words long, separated by a
B90          blank line.  The offset, if being printed by the -offset control, is
B91          reset to the initial value at the beginning of each block.
B92
B93
B94       Control arguments (data format):
B95       -interpreted, -it
B96          prints the data decoded into the indicated format.
B97       -no_interpret, -nit
B98          suppresses printing of the decoded data.  (Default)
B99
B100      -raw
B101         indicates that the raw data is to be printed.  (Default)
B102      -no_raw, -nraw
B103         suppresses printing of the raw data.
B104
B105
Preceding:
```

```
A112      -suppress_duplicates, -sd


A116
A117
A118      Control arguments for structure display:
Changed by B to:
B110      -no_suppress_duplicates, -nsd
B111         indicates that sequential lines are to be printed even if they would
B112         be identical to previous lines.
B113
B114
B115      Control arguments (structure display as a command):
B116         The following control arguments may not be given when dump_segment
B117         is invoked as an active function.


Inserted in B:
B126
B127
Preceding:
A127      -in VIRTUAL_ENTRY


A132
A133
Deleted by B, preceding:
B133      -long, -lg


A141      Control arguments for interpreted data:
Changed by B to:
B140      Control arguments (interpreted data):


A175      Control arguments for raw data:
Changed by B to:
B174      Control arguments (raw data):


A190      -no_offset, -raw, and -supress_duplicates with -header if the entire
Changed by B to:
B189      -no_offset, -raw, and -suppress_duplicates with -header if the entire


A198      If you invoke -4bit, -bcd, -character, -ebcdic8, -ebcdic9, -hex8, or
Changed by B to:
B197      If you give -4bit, -bcd, -character, -ebcdic8, -ebcdic9, -hex8, or


A202      In the active function the following control arguments are invalid--
Changed by B to:
B201      In the active function, the following control arguments are invalid:


A213      Notes for structure display:
Changed by B to:
B212      Notes on structure display:


A253      Examples of structure display:
A254      ds 234 -as stack_header
A255         the whole structure "stack_header".
```

```
A256      ds 234 -as stack_header/ptr/,
A257      ds 234 -as stack_header/ptr
A258         any elements in the structure "stack_header" containing the string
A259         "ptr".  Note that the final slash is optional.
A260      ds 234 -as stack_header/isot/lot/
A261         any elements in the structure "stack_header" containing either the
A262         string "isot" or the string "lot".
A263
A264
A265      ds 234 -as stack_header.stack_begin_ptr
A266         the single element "stack_begin_ptr" in the structure "stack_header".
A267         The following examples assume it has a value of 234|2000.
A268      ds 234|2000 -as stack_frame.pointer_registers
A269         all elements of array "stack_frame.pointer_registers".
A270      ds 234|2000 -as stack_frame.pointer_registers{3}
A271         element three of "stack_frame.pointer_registers".
A272      ds 234|2000 -as stack_frame.pointer_registers{2:4}
A273         elements two, three, and four of "stack_frame.pointer_registers".
A274
A275
A276      ds prog_data -as data_array -in my_prog
A277         all elements in the data_array declared in the my_prog object
A278         program, compiled with -table.  The array is stored at
A279         [wd]>prog_data|0.
A280
A281
A282      Structure output format:
Changed by B to:
B252      Notes on structure output format:


A304           years of the present.
Changed by B to:
B274           years of the present
B275
B276
B277      Examples:
B278      ds 234 -as stack_header
B279         display the whole structure "stack_header" which is stored at the
B280         beginning of segment 234.
B281
B282      ds 234 -as stack_header/ptr/,
B283      ds 234 -as stack_header/ptr
B284         any elements in the structure "stack_header" containing the string
B285         "ptr".  Note that the final slash is optional.
B286
B287      ds 234 -as stack_header/isot/lot/
B288         any elements in the structure "stack_header" containing either the
B289         string "isot" or the string "lot".
B290
B291
B292      ds 234 -as stack_header.stack_begin_ptr
B293         display the single element "stack_begin_ptr" in the structure
B294         "stack_header".  The following examples assume this pointer has a
B295         value of 234|2000...
B296
B297      ds 234|2000 -as stack_frame.pointer_registers
B298         all elements of array "stack_frame.pointer_registers".
B299
B300      ds 234|2000 -as stack_frame.pointer_registers{3}
B301         element three of "stack_frame.pointer_registers".
B302
B303      ds 234|2000 -as stack_frame.pointer_registers{2:4}
```

```
B304         elements two, three, and four of "stack_frame.pointer_registers".
B305
B306
B307      ds prog_data -as data_array -in my_prog
B308         display all elements in the data_array declared in the my_prog
B309         object program, compiled with -table.  The array is stored at
B310         [wd]>prog_data|0.
B311
B312
B313      :hcom:
B314      /****^  HISTORY COMMENTS:
B315        1) change(2026-01-12,GDixon):
B316           Revised for integration with cv_ptr_$for_dump_segment.
B317                                                    END HISTORY COMMENTS */

Comparison finished: 21 differences, 247 lines.
```

Information was added to the info segment for the ring_zero_peek_.info subroutine describing which users can access this subroutine. The changes are shown below.

```
compare_ascii >doc>info>ring_zero_peek_.info ==

A1        02/13/84  ring_zero_peek_
Changed by B to:
B1        :Info: ring_zero_peek_:  2026-05-02  ring_zero_peek_


A4        inner ring segment.  The user must have access to either the phcs_ gate
A5        or the metering_ring_zero_peek_ gate in order to use any of the entry
A6        points in this subroutine.  The phcs_ gate allows unrestricted access
A7        to all inner ring segments; metering_ring_zero_peek_ allows the user to
A8        examine specifically those data bases that are useful for metering the
A9        system.  The program chooses the appropriate gate depending on the
A10       user's access and the segments being examined.
A11
A12
A13       Entry points in ring_zero_peek_:
A14          (List is generated by the help command)
A15
A16
A17       :Entry: ring_zero_peek_: 02/13/84 ring_zero_peek_
Changed by B to:
B4        inner ring segment.  The user must have access to either the >sl1>phcs_
B5        gate or the >sss>metering_gate_ in order to use any of the entry points
B6        in this subroutine.  The phcs_ gate allows unrestricted access to all
B7        inner ring segments; metering_gate_ allows the user to examine
B8        specifically those data bases that are useful for metering the system.
B9        The program chooses the appropriate gate depending on the user's access
B10       and the segments being examined.
B11
B12
B13       :Entry: ring_zero_peek_: 2026-05-02  ring_zero_peek_


A42       :Entry:  by_definition:  02/13/84 ring_zero_peek_$by_definition
Changed by B to:
B38       Access required:  The user must have access to either the >sl1>phcs_
B39       gate or the >sss>metering_gate_ in order to use any of the entry points
B40       in this subroutine.  The phcs_ gate allows unrestricted access to all
B41       inner ring segments; metering_gate_ allows the user to examine
B42       specifically those data bases that are useful for metering the system.
B43       The program chooses the appropriate gate depending on the user's access
B44       and the segments being examined.
B45
B46
B47       :Entry:  by_definition:  2026-05-02 ring_zero_peek_$by_definition


A91       :Entry:  by_name:  02/13/84 ring_zero_peek_$by_name
Changed by B to:
B96       Access required:  The user must have access to either the >sl1>phcs_
B97       gate or the >sss>metering_gate_ in order to use any of the entry points
B98       in this subroutine.  The phcs_ gate allows unrestricted access to all
B99       inner ring segments; metering_gate_ allows the user to examine
B100      specifically those data bases that are useful for metering the system.
B101      The program chooses the appropriate gate depending on the user's access
B102      and the segments being examined.
B103
```

```
B104
B105      :Entry:  by_name:  2026-05-02 ring_zero_peek_$by_name


A135      :Entry:  get_max_length:  02/13/84 ring_zero_peek_$get_max_length
Changed by B to:
B149      Access required:  The user must have access to either the >sl1>phcs_
B150      gate or the >sss>metering_gate_ in order to use any of the entry points
B151      in this subroutine.  The phcs_ gate allows unrestricted access to all
B152      inner ring segments; metering_gate_ allows the user to examine
B153      specifically those data bases that are useful for metering the system.
B154      The program chooses the appropriate gate depending on the user's access
B155      and the segments being examined.
B156
B157
B158      :Entry:  get_max_length:  2026-05-02 ring_zero_peek_$get_max_length


A158      :Entry:  get_max_length_ptr:
A159      02/13/84 ring_zero_peek_$get_max_length_ptr
Changed by B to:
B181      Access required:  The user must have access to either the >sl1>phcs_
B182      gate or the >sss>metering_gate_ in order to use any of the entry points
B183      in this subroutine.  The phcs_ gate allows unrestricted access to all
B184      inner ring segments; metering_gate_ allows the user to examine
B185      specifically those data bases that are useful for metering the system.
B186      The program chooses the appropriate gate depending on the user's access
B187      and the segments being examined.
B188
B189
B190      :Entry:  get_max_length_ptr:
B191      2026-05-02  ring_zero_peek_$get_max_length_ptr


Inserted in B:
B217
B218
B219      Access required:  The user must have access to either the >sl1>phcs_
B220      gate or the >sss>metering_gate_ in order to use any of the entry points
B221      in this subroutine.  The phcs_ gate allows unrestricted access to all
B222      inner ring segments; metering_gate_ allows the user to examine
B223      specifically those data bases that are useful for metering the system.
B224      The program chooses the appropriate gate depending on the user's access
B225      and the segments being examined
B226
B227
B228      :hcom:
B229      /****^  HISTORY COMMENTS:
B230        1) change(2026-05-02,GDixon):
B231            A) Correct name from metering_ring_zero_peek_ gate to
>sss>metering_gate_.
B232            B) Add "Access required" sections to each entry point so they will be
B233               displayed if the user asks help for a specific entry point name.
B234                                                    END HISTORY COMMENTS */
At end

Comparison finished: 7 differences, 90 lines.
```

The changes to ssu_.info are shown as a comparison. They involve documenting the 5 new entry points, plus the 2 existing entry points added some time back without including the documentation.

```
compare_ascii >doc>info>ssu_.info ==

A1        :Info: ssu_:  2025-03-25  ssu_
Changed by B to:
B1        :Info: ssu_:  2025-06-26  ssu_


Inserted in B:
B557      :Entry:  display_tokens:  2025-05-31  ssu_$display_tokens
B558
B559
B560      Function:  This entry displays token descriptors found by the
B561      ssu_$get_tokens subroutine.  See "Notes on token display" for details
B562      on how a token descriptor is displayed.
B563
B564
B565      Syntax:
B566      declare ssu_$display_tokens entry(ptr, char(32) var, ptr,
B567           bit(1) aligned, bit(1) aligned);
B568
B569      call ssu_$display_tokens(sci_ptr, input_string_name, first_token_ptr,
B570           list_display_sw, statement_display_sw);
B571
B572
B573      Arguments:
B574      sci_ptr
B575         is a pointer to the subsystem control structure for this invocation
B576         returned by either ssu_$create_invocation or
B577         ssu_$standalone_invocation.  (Input)
B578      input_string_name
B579         is a string describing the input string which included the lexemes
B580         described in the displayed tokens.  (Input)
B581      first_token_ptr
B582         is a pointer to the token describing the first lexeme found in the
B583         input string.  (Input)  Sibling tokens are pointed to by the
B584         token.nextP and token.prevP elements.
B585
B586
B587      list_display_sw
B588         is set to "1"b if contents of list-flavor lexemes should be
B589         displayed to identify sub-lexemes found within the string bound by
B590         list start and end characters.  (Input)  Child tokens are pointed to
B591         by the token.childP element of each list-type token descriptor.
B592      statement_display_sw
B593         is set to "1"b to display tokens using a statement-by-statement
B594         ordering if statement descriptors are present for top-level tokens.
B595         If set to "0"b, then tokens are displayed by-level one token at a
B596         time without display of statement contents or attention to statement
B597         delimiter tokens.
B598
B599
```

```
B600      Notes on token display:
B601      Tokens are shown in a hierarchical display.  The top-level tier
B602      displays siblings describing the basic lexemes (meaningful substrings)
B603      found in the original input string.  For a string:
B604
B605         "This is the <input-string>."
B606
B607      the top tier includes tokens numbered from 1 through 5 describing each
B608      lexeme found by the first scan of the input string.  The display below
B609      is produced by giving ssu_$display_tokens an input_string_name =
B610      "Input" and list_display_sw = "0"b.
B611
B612
B613        Input
B614         1: This
B615         2: is
B616         3: the
B617         4: <input-string>  | AngleList hasChild
B618         5: .
B619
B620
B621      The fourth token describes an angle-bracketed-list lexeme.  The
B622      AngleList property denotes a token.flavor differing from the
B623      default value (CharToken) of the earlier tokens in the display.
B624
B625      The hasChild attribute indicates that ssu_$get_tokens was invoked with
B626      an argument list_rescan_sw = "1"b.  This requests that child tokens are
B627      created for sub-lexemes found within each list-type lexeme.  These
B628      child tokens are chained to their parent token using the token.childP
B629      element.
B630
B631
B632      If ssu_$display_tokens is invoked with argument list_display_sw = "1"b,
B633      then any child tokens are also displayed.
B634
B635        Input
B636         1: This
B637         2: is
B638         3: the
B639         4: <input-string>  | AngleList hasChild
B640         5: .
B641
B642         4.1: input-string  | hasParent
B643
B644
B645      Examples:
B646      The following shows display of a more complex input string including
B647      several top-level list-type lexemes.  ssu_$get_tokens was invoked with
B648      list_rescan_sw = "1"b.  ssu_$display_tokens was invoked with
B649      list_display_sw = "1".  For an input string:
B650
B651         "X [bc (a1a2 (a3 (25))) ]  <def Z {ghij}>"
B652
B653      the ssu_$display_tokens output is shown below.
B654
B655
```

```
B656        Input
B657         1: X   2: [bc (a1a2 (a3 (25))) ]  | SquareList hasChild
B658         3: <def Z {ghij}>  | AngleList hasChild
B659
B660         2.1: bc  | hasParent
B661         2.2: (a1a2 (a3 (25)))  | ParenList hasParent hasChild
B662
B663         2.2.1: a1a2  | hasParent
B664         2.2.2: (a3 (25))  | ParenList hasParent hasChild
B665
B666         2.2.2.1: a3  | hasParent
B667         2.2.2.2: (25)  | ParenList hasParent hasChild Int=25
B668
B669         2.2.2.2.1: 25  | hasParent Int=25
B670
B671         3.1: def  | hasParent   3.2: Z  | hasParent
B672         3.3: {ghij}  | BracedList hasParent hasChild
B673
B674         3.3.1: ghij  | hasParent
B675
B676
B677      The bottom tier with token ID 2.2.2.2.1 is a sub-lexeme consisting
B678      only of characters which form a string representation of an
B679      integer:
B680         25
B681      So the numeric value of that integer string is stored in the
B682      token.value element of its descriptor; this numeric value is indicated
B683      by Int=25 in the display.
B684
B685
B686      The preceding tier with token ID 2.2.2.2 shows a token with only a
B687      parenthesized list containing one number:
B688         (25)
B689      Therefore, the ssu_$get_tokens rescan of list-type lexemes has promoted
B690      the integer value from the token ID 2.2.2.2.1:
B691         25
B692      to be the integer value represented by the parenthesized list in token
B693      ID 2.2.2:
B694         (25)
B695
B696
B697      The next example shows the output for an input string containing
B698      two statements with statement_delimiter = ";".  The input_string is:
B699          "positional pos(1);  control(-name -nm) optional;"
B700
B701      Eight top-level tokens are displayed in the two statements.
B702
B703
B704      "positional pos(1);"                  Input in STATEMENT 1 ON LINE   1
B705
B706         1: positional   2: pos   3: (1)  | ParenList hasChild int=1
B707         4: ;  | endStmt
B708
B709         3.1: 1  | hasParent int=1
B710
B711
B712      "control(-name -nm) optional;"        Input in STATEMENT 2 ON LINE   1
B713
B714         5: control   6: (-name -nm)  | ParenList hasChild   7: optional
B715         8: ;  | endStmt
B716
B717         6.1: -name  | hasParent   6.2: -nm  | hasParent
B718
```

```
B719
Preceding:
A557      :Entry:  evaluate_active_string:  1983-01-19  ssu_$evaluate_active_string


Inserted in B:
B892
B893
Preceding:
A729      code


A738
A739
Deleted by B, preceding:
B903         ssu_et_$subsystem_aborted


Inserted in B:
B907
Preceding:
A744      optional_ec_args


Inserted in B:
B957      :Entry:  find_token_with_id:  2025-05-31  ssu_$find_token_with_id
B958
B959
B960      Function:  Given a pointer to the first token in a token tree returned
B961      by ssu_$get_tokens and a token ID locating some token within that tree,
B962      this subroutine returns a pointer to the token having that ID; or
B963      returns a null pointer with non-zero status code if no token matches
B964      that ID.
B965
B966
B967      Syntax:
B968      declare ssu_$find_token_with_id (ptr, ptr, char(52) var, ptr,
B969                                       fixed bin(35));
B970
B971      call ssu_$find_token_with_id (sci_ptr, first_token_ptr, token_id,
B972                                    found_token_ptr, code);
B973
B974
B975      Arguments:
B976      sci_ptr
B977         is a pointer to the subsystem control structure for this invocation
B978         returned by either ssu_$create_invocation or
B979         ssu_$standalone_invocation.  (Input)
B980      first_token_ptr
B981         is a pointer to the first token in the token tree to be searched.
B982         This must be a token tree returned by the ssu_$get_tokens
B983         subroutine.
B984      token_id
B985         is an identifier locating a token structure within that token tree.
B986         For more information, see "Notes on the token identifier" below.
B987      found_token_ptr
B988         is a pointer to the token whose location in the token tree matches
B989         token_id.
B990
B991
B992      code
B993         is a Multics status code indicating the type of error encountered by
B994         the find operation.  found_token_ptr will be null is this code has a
```

```
B995          non-zero value.  Possible values are:
B996
B997            error_table_$badarg    Improperly formatted token_id argument.
B998            error_table_$bigarg    More than 10 nibbles found in the token_id
B999                                   argument.
B1000           error_table_$noentry   Matching token was not found in token tree.
B1001
B1002
B1003       Notes on the token identifier:
B1004       Given a pointer to any token, ssu_$token_id returns the value of its
B1005       token identifier as a character string with one or more nibbles.  For
B1006       example: 3.1
B1007
B1008       The example below shows a token tree returned by ssu_$get_tokens for
B1009       the string:
B1010
B1011            "a bc (def lmn) ghi[43]"
B1012
B1013       Each token describes a lexeme found in this input string by
B1014       ssu_$get_tokens.  Token ID 3.1 identifies the token describing the
B1015       lexeme: def
B1016
B1017
B1018            first_token_ptr
B1019              |
B1020              |     token
B1021              |      .nextP    .nextP          .nextP    .nextP
B1022               -->  _   -->  __   -->  _________  -->  ___   -->  ____
B1023                    a       bc       (def lmn)       ghi       [43]
B1024                    -       --       ---------       ---       ----
B1025                                         | .childP                | .childP
B1026                                         |                        |
B1027                                         |        .nextP          -->  __
B1028                                         -->  ___  -->  ___             43
B1029                  ID 3.1 locates ..... def       lmn             --
B1030                                        ---       ---
B1031
B1032
B1033       ssu_$get_tokens returns a first_token_ptr argument which points to the
B1034       first of 5 top-level token structures.  token.nextP of each token
B1035       points to next sibling token found by ssu_$get_tokens.  token.I gives
B1036       the order in which that token appears on  that branch of the tree.
B1037
B1038       If the list_rescan_sw argument was set to "1"b, then each token
B1039       describing a list-lexeme (e.g., token.flavor = ParenList) has a
B1040       token.childP value pointing to a subsidiary list of tokens describing
B1041       lexemes found within the list-lexeme string.  The parenthesized list
B1042       "(def lmn)" holds two sub-lexemes: "def" and "lmn" on its token.childP
B1043       branch.  The bracketed list "[43]" holds one sub-lexeme: "43" on its
B1044       own token.childP branch.
B1045
B1046
B1047       ssu_$display_tokens displays information about each token in the token
B1048       tree beginning with its token identifier, followed by its tokenStr
B1049       value (the sub-string which is the lexeme), and any special properties
B1050       and attributes of that lexeme.
B1051
B1052
B1053            Tokens by ID
B1054             1: a                                    \
B1055             2: bc                                    \
B1056             3: (def lmn)  | ParenList hasChild        >   TOP-LEVEL TOKENS
B1057             4: ghi                                   /
```

```
B1058           5: [43]  | SquareList hasChild Int=43  /
B1059
B1060           3.1: def  | hasParent                  \     CHILD TOKENS of
B1061           3.2: lmn  | hasParent                  /      token ID: 3
B1062
B1063           5.1: 43  | hasParent Int=43            >     CHILD TOKEN of
B1064                                                         token ID: 5
B1065
B1066
B1067     For each token, its token.I value gives an ID nibble which locates
B1068     that token on its child branch of the token tree: token.I = 1 for first
B1069     token, ..., token.I = n for the nTH token.
B1070
B1071     Each token with non-null token.childP introduces a new nibble in the
B1072     identifier.  For example, an ID = 3.1 starts with the third TOP-LEVEL
B1073     token, then selects its first child token: the token describing the
B1074     "def" lexeme in the tree shown above.
B1075
B1076
Preceding:
A793      :Entry:  get_area:  1983-01-19  ssu_$get_area


A798      an area in a temporary segment.  The difference between using this
A799      entry and calling define_area_ directly is that areas acquired by
A800      calling ssu_$get_area are released when the subsystem invocation is
A801      destroyed, regardless of whether the user program had freed them
A802      earlier.  Areas acquired by calling ssu_$get_area should be released by
A803      calling ssu_$release_area.
Changed by B to:
B1082     an area in a temporary segment managed by ssu_.  Areas acquired by
B1083     calling ssu_$get_area may be released by calling ssu_$release_area.
B1084     They will be released automatically when the subsystem invocation is
B1085     destroyed.
B1086
B1087     See also the ssu_$get_token_area function which returns an extensible,
B1088     no-freeing area in an SSU temporary segment.  Such area is ideal for
B1089     holding tokens created by the ssu_$get_tokens subroutine.


A818         area_info.incl.pl1) describing the area, or null.  (Input) If the
A819         area_info_ptr is null, an area with default characteristics is
A820         defined; see below for details.  Only the area_control flags in the
A821         area_info are used by ssu_$get_area; the area is always put in a
A822         temporary segment.  The area pointer is returned as the final
A823         argument to ssu_$get_area; it is not written into the area_info
A824         structure.
Changed by B to:
B1104        area_info.incl.pl1) describing the area.  (Input) If the
B1105        area_info_ptr is null, an area with default characteristics is
B1106        defined; see "Notes" below for details.  Only the area_info.control
B1107        flags are referenced by ssu_$get_area.  In particular, the
B1108        area_info.areap pointer element remains unset by this call; see the
B1109        area_ptr argument below.


A845      with default characteristics is created.  The area is extensible,
A846      initially one segment long, and is zero_on_free (but not
A847      zero_on_alloc).  All other area_control flags are off.
Changed by B to:
B1130     with default characteristics is created: an area which is extensible,
B1131     initially one segment long, having the zero_on_free flag set.  All
B1132     other area_info.control flags are off.
```

```
Inserted in B:
B1746     :Entry:  get_token_area:  2025-05-31 ssu_$get_token_area
B1747
B1748
B1749     Function: This subroutine obtains an area with characteristics ideal
B1750     for tokens created by the ssu_$get_tokens subroutine.  It calls the
B1751     ssu_$get_area subroutine to obtain an extensible, no-freeing area in a
B1752     temporary segment managed by ssu_.  The ssu_ request or standalone
B1753     command may explicitly release the area by calling ssu_$release_area;
B1754     or it will be released automatically when the subsystem invocation is
B1755     destroyed.
B1756
B1757
B1758     Syntax:
B1759     declare ssu_$get_token_area entry (ptr, char(*)) returns (ptr);
B1760
B1761     token_area_ptr = ssu_$get_token_area (sci_ptr, comment);
B1762
B1763
B1764     Arguments:
B1765     token_area_ptr
B1766        is a pointer to the area.  (Output) It will always be valid; if for
B1767        some reason the area cannot be acquired, the current request line
B1768        (or subsystem invocation, if there is no request line) is aborted
B1769        with an appropriate message.  No errors are ever reflected back to
B1770        the caller of ssu_$get_token_area.
B1771     sci_ptr
B1772        is a pointer to the subsystem control structure for this invocation
B1773        as returned by ssu_$create_invocation.  (Input)
B1774
B1775
B1776     comment
B1777        is a comment identifying the use to which the area will be put.
B1778        (Input) It is used in constructing the owner name for the call to
B1779        define_area_, in the form:
B1780                subsys_name.N (comment)
B1781        where subsys_name is the name of the subsystem, N is the invocation
B1782        level for this invocation, and the comment (if any) follows, in
B1783        parentheses.  This is done to make it easier to identify the segment
B1784        names listed by list_temp_segments.
B1785
B1786
B1787     :Entry:  get_tokens:  2025-06-11  ssu_$get_tokens
B1788
B1789
B1790     Function:  This entry is called to find lexemes (meaningful substrings)
B1791     within an input string.  A token descriptor is allocated to describe
B1792     each lexeme found in the input_string.  Descriptors are linked in a
B1793     list in the order of appearance of their lexeme within the input
B1794     string.  See "Notes on lexemes" for rules on how lexemes are identified
B1795     in the input string.
B1796
B1797     You may use the ssu_$display_tokens subroutine to display the tokens
B1798     returned by ssu_$get_tokens.
B1799
B1800
B1801     Syntax:
B1802     declare ssu_$get_tokens entry(ptr, ptr,
B1803          char(*), fixed bin(21), char(32) var,
B1804          ptr, bit(1) aligned,
B1805          char(4) var,  char(*) var,
```

```
B1806          ptr, fixed bin, fixed bin(35));
B1807
B1808     call ssu_$get_tokens(sci_ptr, token_area_ptr,
B1809          input_string, ignored_inputL, input_string_name,
B1810          caller_data_ptr, list_rescan_sw,
B1811          statement_delimiter, extra_break_chars,
B1812          first_token_ptr, token_count, code);
B1813
B1814
B1815     Arguments:
B1816     sci_ptr
B1817        is a pointer to the subsystem control structure for this invocation
B1818        returned by either ssu_$create_invocation or
B1819        ssu_$standalone_invocation.  (Input)
B1820     token_area_ptr
B1821        is a pointer to an area in which tokens should be allocated.
B1822        (Input)  See "Notes on token area" below for further details.
B1823
B1824
B1825     input_string
B1826        is a string to be searched for lexemes.  (Input)  See "Notes on
B1827        lexemes" below to learn about the rules for identifying lexemes
B1828        within the input_string.
B1829     ignored_inputL
B1830        is the length (in characters) of the first section of the input
B1831        string, text that is ignored except for counting lines within the
B1832        ignored text.  (Input).  Set this length to 0 if none of the input
B1833        characters are to be ignored.
B1834     input_string_name
B1835        is a string identifying the nature of the input in messages
B1836        displaying lexemes found, or diagnosing problems with the search.
B1837        (Input)
B1838
B1839
B1840     caller_data_ptr
B1841        is a pointer to caller-specified data.  This pointer is copied into
B1842        the token.callerP element of each token descriptor.  (Input)
B1843     list_rescan_sw
B1844        is set to "1"b if contents of list-type lexemes should be rescanned
B1845        to identify sub-lexemes found within the string bound by list start
B1846        and end characters.  (Input)  The rescan excludes those start and
B1847        end characters.  If set to "0"b, then these extra scans are not
B1848        performed.  The token.childP element points to the token descriptor
B1849        for the first sub-lexeme found in the contents of each list-type
B1850        lexeme.  Sibling tokens are pointed to by the token.nextP and
B1851        token.prevP elements.
B1852
B1853
B1854     statement_delimiter
B1855        a special character string (lexeme) that ends each statement
B1856        (sequence of character strings) contained in the input_string.
B1857        (Input)  A semi-colon is often used as the statement_delimiter.  But
B1858        the caller may choose any string of 1 to 4 characters selected from
B1859        these characters:
B1860
B1861                ampersand (&)          dollar-sign ($)      semi-colon (;)
B1862                 asterisk (*)    exclamation-point (!)    single-quote (')
B1863                  at-sign (@)        forward-slash (/)            tilde (~)
B1864           backward-quote (`)               period (.)    vertical-bar (|)
B1865           backward-slash (\)           pound-sign (#)
B1866               circumflex (^)        question-mark (?)
B1867
B1868
```

```
B1869        If it is a zero-length string, then statement descriptor structures
B1870        are not produced while identifying lexemes in the input_string.  For
B1871        more information, see "Notes on lexemes" below.
B1872
B1873
B1874     extra_break_chars
B1875        one or more characters to be treated as a "break character" that
B1876        ends the formation of a lexeme.  Each becomes a 1-character lexeme
B1877        itself.
B1878
B1879
B1880     first_token_ptr
B1881        is a pointer to the token describing the first lexeme found in the
B1882        input_string.  (Output)  See "Notes on tokens" for more details
B1883        about these descriptors.  Sibling tokens are pointed to by the
B1884        token.nextP element.
B1885     token_count
B1886        is a count of the lexeme token descriptors found within the
B1887        input_string.  (Output)  Descriptors for sub-lexemes are not
B1888        included in this token_count.
B1889     code
B1890        is a Multics status code.  (Output)
B1891
B1892
B1893     Notes on lexemes:  The input string is scanned for lexemes (important
B1894     characters within the input_string) using the following rules.
B1895
B1896     Scan characters of the input string from left-to-right.  Whitespace
B1897     characters are generally treated as token separators which are not
B1898     contained within a lexeme.  A lexeme may then be defined as follows.
B1899
B1900
B1901     lexeme
B1902        is a sequence of characters within the input string which:
B1903         - begins with the first non-whitespace character found by the
B1904           left-to-right scan.
B1905         - ends with the first whitespace character, break character,
B1906           double-quote or list-starting character found during the scan.
B1907
B1908        whitespace
B1909           are characters which separate lexemes from one another.  These
B1910           characters are often not included within any lexeme.  Whitespace
B1911           characters include:
B1912              horizontal_tab (HT)   space (SP)   newline (NL)   newpage (NP)
B1913
B1914
B1915        break characters
B1916           are characters which break or stop the scan to find the end of
B1917           the current lexeme.  The current lexeme ends with the character
B1918           preceding each of these break characters.  The break character
B1919           itself is then treated as the next lexeme, which is a 1-character
B1920           lexeme.  The break characters include:
B1921              comma (,)   colon (:)   equal-sign (=)   semi-colon (;)
B1922
B1923
B1924     statement
B1925        a sequence of lexemes ended by a statement_delimiter lexeme.  A
B1926        statement descriptor (stmt structure) points to the first and last
B1927        tokens of a statement, and a stmtStr character string overlays
B1928        the part of input_string contained within the statement.
B1929
B1930
B1931     statement_delimiter
```

```
B1932        a special lexeme that ends a statement.  A semi-colon is often used
B1933        as the statement_delimiter.  But the caller may choose any string of
B1934        1 to 4 characters selected from the character-set:
B1935
B1936             ampersand (&)         dollar-sign ($)     semi-colon (;)
B1937              asterisk (*)   exclamation-point (!)   single-quote (')
B1938               at-sign (@)       forward-slash (/)          tilde (~)
B1939        backward-quote (`)             period (.)   vertical-bar (|)
B1940        backward-slash (\)           pound-sign (#)
B1941            circumflex (^)       question-mark (?)
B1942
B1943        The statement_delimiter lexeme is treated as a break string that
B1944        ends any preceding lexeme.  Therefore, the delimiter should not
B1945        include characters expected to appear within other lexemes.
B1946
B1947
B1948        If the statement_delimiter argument is a zero-length (empty string),
B1949        then statements within the input_string are not identified and no
B1950        stmt structures are produced.
B1951
B1952
B1953     Certain characters used to start a lexeme begin an extended lexeme that
B1954     may include whitespace or other characters usually treated specially.
B1955
B1956
B1957     quoted string
B1958        is a sequence of characters which start and end with a double-quote
B1959        (") character.  All characters between and including the starting
B1960        and ending double-quote are treated as the quoted string.  Two kinds
B1961        of quoted strings are supported:
B1962           quoted character string,  quoted bit string
B1963
B1964
B1965        quoted character string lexeme
B1966           is a string of characters starting and ending with a double-quote
B1967           (") character.  Any character may appear between the starting and
B1968           ending double-quote.  A pair of consecutive double-quote
B1969           characters ("") represents one double-quote appearing within the
B1970           quoted character string.
B1971              Example:  "A single ""quoted-character"" string."
B1972           Each quoted character string token has a token.childP pointing
B1973           to a token descriptor for its contents: child_tokenStr which is
B1974           an unquoted string with the surrounding double-quote characters
B1975           removed; and with any escape double-quote pairs reduced to a
B1976           single double-quote char.  child_token.flavor = CharToken.
B1977
B1978
B1979        quoted bit string lexeme
B1980           is a string of characters starting with a double-quote (")
B1981           character and ending with a double-quote character immediately
B1982           followed by one or two bit-string indicator characters:
B1983
B1984              INDICATOR    CHARACTERS BETWEEN DOUBLE-QUOTES ARE
B1985              "b or "b1     binary digits         0 1
B1986                    "b2     quaternary digits     0 1 2 3
B1987                    "b3     octal digits          0 1 2 3 4 5 6 7
B1988                    "b4     hexadecimal digits    0 1 2 3 4 5 6 7 8 9
B1989                                                  A B C D E F a b c d e f
B1990              Examples:
B1991               "011"b1                           (bit string of length 3)
B1992               "753"b3   =  "111101011"b         (bit string of length 9)
B1993               "1A3"b4   =  "000110100011"b      (bit string of length 12)
B1994
```

```
B1995
B1996     list lexeme
B1997        is a sequence of characters within the input string which:
B1998           - begins with a particular starting character, and
B1999           - ends with a particular ending character.
B2000
B2001        Unlike a quoted character string lexeme, contents of a list lexeme
B2002        (excluding its start and end characters) may be scanned again if
B2003        list_rescan_sw = "1"b to look for sub-lexemes contained within
B2004        the list.  Sub-lexemes found in the list will be treated as
B2005        child tokens of the list lexeme.  Four list kinds are supported:
B2006           square-bracketed  parenthesized  braced  angle-bracketed
B2007
B2008
B2009        square-bracketed-list lexeme
B2010           is a list of characters starting with a left-bracket ([)
B2011           character and ending with the first subsequent right-bracket (])
B2012           character.  All characters between and including the start and
B2013           end brackets are treated as part of the lexeme.
B2014
B2015           A square-bracketed-list may not be nested within an outer
B2016           square-bracketed-list.  It may include several left-bracket
B2017           characters, but ends with the first right-bracket character
B2018           encountered by the scan.  Example:  [x + 42 = 50]
B2019
B2020
B2021        nested-list lexemes
B2022           If a list begins with one of the nested-list starting characters,
B2023           then it ends as follows.
B2024            - Set nesting count to 1.
B2025            - Scan for either the start character of this list kind or
B2026              its list end character.
B2027               - If another start character is found, increase the
B2028                 nesting count by 1.
B2029               - If the list's end character is found, decrease the
B2030                 nesting count by 1, and continue scanning for end-of-list
B2031                 until the nesting count equals 0.
B2032
B2033           This permits a nested list to include any character.  Both its
B2034           starting and ending character are included in the lexeme.
B2035
B2036
B2037           The three nested-list kinds are as follows.
B2038
```

```
B2039           parenthesized-list lexeme
B2040              a list of characters starting with a left-parenthesis (() and
B2041              ending with a right-parenthesis ()) character.
B2042              Examples:   (a + b)   (45.2)   (5 + (a * b) - c)
B2043
B2044           braced-list lexeme
B2045              a list of characters starting with a left-brace ({) and ending
B2046              with a right-brace (}).  Example:   {abc {def ghi} jkl}
B2047
B2048           angle-bracketed-list lexeme
B2049              a list of characters starting with a less-than (<) and ending
B2050              with a greater-than (>) character.
B2051              Example:   <statement-kinds>
B2052
B2053
B2054       Notes on tokens: The ssu_$get_tokens subroutine creates a token
B2055       describing each lexeme found while scanning the input_string argument.
B2056       Each token is allocated in the area located by the token_area_ptr
B2057       argument.  The following declaration from the ssu_token.incl.pl1
B2058       include segment shows the information provided for each lexeme.
B2059
B2060
B2061         dcl 1 token                  aligned based (tokenP),
B2062               2 group1               unaligned,
B2063                 3 callerP            ptr unal,      /* caller's data loc  */
B2064               2 nextP                ptr unal,      /* next token loc     */
B2065               2 prevP                ptr unal,      /* prev token loc     */
B2066               2 P                    ptr unal,      /* lexeme tokenStr loc*/
B2067               2 L                    fixed bin(21), /* lexeme tokenStr len*/
B2068               2 parentP              ptr unal,      /* parent token loc   */
B2069               2 childP               ptr unal,      /* 1st child token loc*/
B2070               2 group2 unaligned,
B2071                 3 I                  fixed bin(17), /* token pos in input */
B2072                 3 flavor             fixed bin(17), /* kind of token      */
B2073                 3 value              fixed bin(35), /* integer token value*/
B2074
B2075
B2076                 3 S unaligned,
B2077                   4 end_of_stmt      bit(1),        /* statement_delimiter*/
B2078                   4 quoted_string    bit(1),        /* quoted-string token*/
B2079                   4 quotes_in_string bit(1),        /* has double-quotes  */
B2080                   4 quotes_escaped   bit(1),        /* each double-quote  */
B2081                                                     /* in tokenStr appears*/
B2082                                                     /* as 2 double-quotes.*/
B2083                   4 integer_string   bit(1),        /* integer token value*/
B2084                   4 top_level        bit(1),        /* top of token tree  */
B2085                   4 pad2             bit(30),
B2086               tokenP                 ptr,
B2087               tokenStr               char(token.L) based(token.P);
B2088
B2089
B2090         dcl (                                    /* token.flavor values    */
B2091              CharToken               init(0),
B2092              CharString              init(1),   /*  "quoted char string"  */
B2093              BitString               init(2),   /*  "101"b                */
B2094              SquareList              init(3),   /*  [square-bracket list] */
B2095              ParensList              init(4),   /*  (parenthesized list)  */
B2096              BracedList              init(5),   /*      {braced list}     */
B2097              AngleList               init(6)    /*  <angle-bracket list>  */
B2098                                      fixed bin int static options(constant);
B2099
B2100
B2101       List of token elements:
```

```
B2102        Elements of the token structure in "Notes on tokens" above.  To
B2103        examine a given token structure, assign location of that structure
B2104        to tokenP.  Then the lexeme being described may be accessed using
B2105        the tokenStr declaration.
B2106
B2107     callerP
B2108        is set to the value of the caller_data_ptr argument provided in the
B2109        ssu_$get_tokens subroutine call.
B2110     nextP
B2111        is a pointer to the token structure which describes the next lexeme
B2112        found in the input_string.
B2113     prevP
B2114        is a pointer to the token structure which describes the prior lexeme
B2115        found in the input_string.
B2116
B2117
B2118     P
B2119        is a pointer to the tokenStr character string which contains all
B2120        characters of the lexeme described by this token structure.
B2121     L
B2122        is the length in characters of the lexeme described by this token
B2123        structure.
B2124     parentP
B2125        is usually a null pointer.  However, if list_rescan_sw = "1"b in the
B2126        ssu_$get_tokens call, parentP points to the token structure
B2127        describing the parent list which contains the sub-lexeme described
B2128        by the current token structure.
B2129
B2130
B2131     childP
B2132        is a pointer to the first "child" lexeme found when the current
B2133        token structure describes a list-lexeme and that list tokenStr has
B2134        been rescanned.
B2135     I
B2136        is the order number in which the lexeme described by this token
B2137        structure appears in the input_string.  For tokens found in a parent
B2138        list which has been rescanned, I gives order of appearance of the
B2139        child lexeme within that parent list.
B2140     flavor
B2141        is an integer value representing the kind of lexeme described by
B2142        this token.  Flavors range from 0 through 6 as described by the
B2143        token.flavor constants listed above.
B2144
B2145
B2146     value
B2147        if characters described by the current token appear to represent
B2148        digits of a possibly-signed decimal integer string (with optional
B2149        radix indicator substring), this element holds the numeric value of
B2150        that lexeme as interpreted by the cv_dec_check_ subroutine.
B2151        token.S.integer_string is set to "1"b in such case.  Otherwise,
B2152        value is set to 0.
B2153
B2154
B2155     end_of_stmt
B2156        if set to "1"b if token describes a statement_delimiter lexeme.
B2157     quoted_string
B2158        is set to "1"b if token describes a quoted-character-string lexeme
B2159        or a quoted-bit-string lexeme.
```

```
B2160     quotes_in_string
B2161        is set to "1"b if token describes a quoted-character-string lexeme
B2162        that contains one or more double-quote characters (as a pair of
B2163        double-quote characters in the input_string).
B2164
B2165
B2166     quotes_escaped
B2167        is set to "1"b if tokenStr still has any double-quote characters
B2168        represented by a pair of double-quote characters in the tokenStr
B2169        string pointed to by the token descriptor.
B2170     integer_string
B2171        is set to "1"b if the characters described by the current token
B2172        appear to represent digits of a possibly-signed decimal integer
B2173        string (with optional radix indicator substring) as interpreted by
B2174        the cv_dec_check_ subroutine.  token.value contains the interpreted
B2175        value of that token string.
B2176
B2177
B2178     top_level
B2179        is set to "1"b if this token describes a lememe of the input string
B2180        rather than a child sub-lexeme of a list-flavor lexeme.  For
B2181        top-level tokens, token.parentP points to the stmt descriptor
B2182        describing its statement; or is a null pointer if the
B2183        statement_delimiter argument was an empty string.
B2184
B2185
B2186     Notes on statements:
B2187     If a non-empty statement_delimiter argument is given, then
B2188     ssu_$get_tokens creates a stmt (statement descriptor) structure
B2189     identifying the top-level tokens found in each statement found in the
B2190     input_string.  A statement descriptor can be used to include content of
B2191     the entire statement containing an error found parsing the
B2192     input_string.  Elements of a stmt structure are shown below, and
B2193     included in the ssu_token.incl.pl1 include segment.
B2194
B2195
B2196       dcl 1 stmt                  aligned based (Pstmt),
B2197             2 group1              unaligned,
B2198               3 callerP           ptr unal,         /* caller's data   */
B2199             2 nextP               ptr unal,         /* next stmt loc   */
B2200             2 prevP               ptr unal,         /* prev stmt loc   */
B2201             2 P                   ptr unal,         /* stmt_value loc  */
B2202             2 L                   fixed bin(18),    /* stmt_value len  */
B2203             2 first_tokenP        ptr               /* 1st  token loc  */
B2204             2 last_tokenP         ptr unal,         /* last token loc  */
B2205             2 pad1                (2) fixed bin(35),
B2206             2 group2              unaligned,
B2207               3 tokensN           fixed bin(17) uns,/* tokens in stmt. */
B2208               3 lineI             fixed bin(17) uns,/* stmt line number*/
B2209               3 position_in_lineJ fixed bin(17) uns,/* pos in that line*/
B2210               3 semant_type       fixed bin(17),    /* set by caller   */
B2211
B2212
B2213               3 S,
B2214                 4 error_in_stmt   bit(1),           /* syntax error    */
B2215                 4 output_in_error bit(1),           /* stmt seen before*/
B2216                 4 text_has_NLs    bit(1),           /* NL chars in stmt*/
B2217                 4 pad2            bit(33),
B2218             2 pad3                fixed bin(35),
B2219
B2220             stmtStr               char(stmt.L) based (stmt.P),
B2221                                                     /* Text of stmt    */
B2222             stmtP                 ptr,              /* Pointer to stmt */
```

```
B2223             stmt_delim             char(4) var;      /* stmt delimiter  */
B2224
B2225
B2226     Notes on token area:
B2227     A token describing each lexeme found in the input_string is allocated
B2228     within the area located by token_area_ptr.  Any standard Multics area
B2229     may be used.  These tokens are needed only while interpreting the
B2230     meaning of lexemes in the input_string.  It is therefore useful to
B2231     allocate them in a temp segment containing an extensible, no-freeing
B2232     area which may be released quickly.  With this approach, it is not
B2233     necessary to free tokens allocated in the area.
B2234
B2235
B2236     An extensible, no-freeing area is ideal for tokens.  One may be
B2237     created by calling the ssu_$get_token_area subroutine.
B2238
B2239
B2240     When the token_area is no longer needed, a token area created by
B2241     ssu_$get_area or ssu_$get_token_area  may be released by calling the
B2242     ssu_$release_area subroutine.
B2243
B2244
B2245     Examples:
B2246     The example below calls ssu_$get_tokens to parse the string:
B2247
B2248          "a bc (def lmn) ghi[43]"
B2249
B2250     into the following tree of tokens.  Each token describes a lexeme found
B2251     by ssu_$get_tokens in this input string.
B2252
B2253
B2254          first_token_ptr
B2255            |
B2256            |     token
B2257            |      .nextP    .nextP          .nextP    .nextP
B2258             -->  _   -->  __   -->  _________  -->  ___  -->  ____
B2259                  a       bc       (def lmn)       ghi       [43]
B2260                  -       --       ---------       ---       ----
B2261                                       | .childP                | .childP
B2262                                       |                        |
B2263                                       |        .nextP          -->  __
B2264                                       -->  ___  -->  ___             43
B2265                                            def       lmn            --
B2266                                            ---       ---
B2267
B2268
B2269     ssu_$get_tokens returns a first_token_ptr argument which points to the
B2270     first of 5 top-level token structures.  token.nextP of each token
B2271     points to next token found by ssu_$get_tokens.  If the list_rescan_sw
B2272     argument was set to "1"b, then each token describing a list-lexeme
B2273     (e.g., token.flavor = ParenList) has a token.childP value pointing to a
B2274     subsidiary list of tokens describing lexemes found within the
B2275     list-lexeme string.
B2276
B2277
B2278     ssu_$display_tokens displays information about each token in the token
B2279     tree.  For each token, its token.I value gives an ID nibble which
B2280     locates that token relative to siblings at the same level of the token
B2281     tree; token.I = 1 for first token, ..., token.I = n for the nTH token.
B2282
B2283     Each token with non-null token.childP introduces a new nibble in the
B2284     identifier.  For example, an ID = 3.1 starts with the third TOP-LEVEL
B2285     token, then selects its first child token: the token describing the
```

```
B2286     "def" lexeme in the tree shown above.
B2287
B2288
B2289          Tokens by ID
B2290           1: a                                    \
B2291           2: bc                                    \
B2292           3: (def lmn)  | ParenList hasChild       >   TOP-LEVEL TOKENS
B2293           4: ghi                                   /
B2294           5: [43]  | SquareList hasChild int=43   /
B2295
B2296           3.1: def  | hasParent                   \     CHILD TOKENS of
B2297           3.2: lmn  | hasParent                   /      token ID: 3
B2298
B2299           5.1: 43  | hasParent int=43             >     CHILD TOKEN of
B2300                                                          token ID: 5
B2301
B2302
B2303     The next example shows the output for an input_string containing
B2304     two statements with statement_delimiter = ";".  The input_string is:
B2305         "positional pos(1);  control(-name -nm) optional;"
B2306
B2307     Eight top-level tokens are displayed in the two statements.
B2308
B2309
B2310     "positional pos(1);"                  Input in STATEMENT 1 ON LINE   1
B2311
B2312        1: positional   2: pos   3: (1)  | ParenList hasChild int=1
B2313        4: ;  | endStmt
B2314
B2315        3.1: 1  | hasParent int=1
B2316
B2317
B2318     "control(-name -nm) optional;"        Input in STATEMENT 2 ON LINE   1
B2319
B2320        5: control   6: (-name -nm)  | ParenList hasChild   7: optional
B2321        8: ;  | endStmt
B2322
B2323        6.1: -name  | hasParent   6.2: -nm  | hasParent
B2324
B2325
B2326     A final example shows a child token created for each quoted character
B2327     string token.
B2328
B2329     "String holds a ""quoted-string with quote-chars (""""")"".""
B2330                                                          Input ON LINE 1
B2331
B2332        1: String   2: holds   3: a
B2333        4: "quoted-string with quote-chars ("")"
B2334                        | CharString quotesInString quotesPaired hasChild
B2335        5: .
B2336
B2337        4.1: quoted-string with quote-chars (")  | quotesInString hasParent
B2338
B2339
```

```
B2340     :Entry:  invoke_request:  2025-05-31  ssu_$invoke_request
B2341
B2342
B2343     Function:  This entry is called to locate and invoke a subsystem
B2344     request referenced in a given request line.  Most implementations call
B2345     ssu_$locate_request to convert the given request name into a request
B2346     entry point.
B2347
B2348
B2349     Syntax:
B2350     declare ssu_$invoke_request entry (ptr, char(*), ptr, fixed bin(35));
B2351
B2352     call ssu_$invoke_request (sci_ptr, request_name, arg_list_ptr, code);
B2353
B2354
B2355     Arguments:
B2356     sci_ptr
B2357        is a pointer to the subsystem control structure for this invocation
B2358        as returned by ssu_$create_invocation.  (Input)
B2359     request_name
B2360        is the name of the request to be executed.  (Input)
B2361     arg_list_ptr
B2362        is a pointer to the request argument list.  (Input)
B2363     code
B2364        is a Multics status code.  (Output)
B2365
B2366
B2367     Notes: This is a replaceable procedure.  See the Programmer's
B2368     Reference Manual for information on the use of replaceable procedure
B2369     within interactive subsystems.  The ssu_execute_$execute_request
B2370     subroutine implements this function by default.
B2371
B2372
Preceding:
A1461     :Entry:  list_info_dirs:  2025-03-25  ssu_$list_info_dirs


Inserted in B:
B2579     :Entry:  locate_request:  2025-05-31  ssu_$locate_request
B2580
B2581
B2582     Function:  This entry is called to locate a subsystem request entry
B2583     point given one of its request names.  It returns the full set of
B2584     request names, its request flags (attributes of the request), and an
B2585     entry variable locating the request entry point.
B2586
B2587
B2588     Syntax:
B2589     declare ssu_$locate_request entry (ptr, char(*), ptr, fixed bin(35));
B2590
B2591     call ssu_$locate_request (sci_ptr, request_name, request_data_ptr,
B2592          code);
B2593
B2594
B2595     Arguments:
B2596     sci_ptr
B2597        is a pointer to the subsystem control structure for this invocation
B2598        as returned by ssu_$create_invocation.  (Input)
B2599     request_name
B2600        is the name of the request to be executed.  (Input)
```

```
B2601     request_data_ptr
B2602        is a pointer to the request_data structure.  (Input)  The caller
B2603        provides storage for this structure.  This entry stores values in
B2604        structure elements.  For more information, see "Notes on the
B2605        request_data structure" below.
B2606     code
B2607        is a Multics status code.  (Output)
B2608
B2609
B2610     Notes on the request_data structure:  The _ssu_request_data.incl.pl1
B2611     include segment declares the request_data input structure.  This entry
B2612     fills in elements describing the located request for use by the caller.
B2613     Null data is returned for the request_data.call_info substructure
B2614     elements.
B2615
B2616
B2617        dcl  1 request_data           aligned based (request_data_ptr),
B2618               2 full_name            character (32) unaligned,
B2619               2 entry                entry (pointer, pointer) variable,
B2620               2 call_info,
B2621                 3 arg_list_ptr       pointer,
B2622                 3 arg_count          fixed binary,
B2623                 3 af_sw              bit (1) aligned,
B2624                 3 rv_ptr             pointer,
B2625                 3 rv_lth             fixed binary (21),
B2626               2 flags                aligned like request_flags,
B2627               2 name_list_ptr        pointer unaligned,
B2628               2 info_string,
B2629                 3 ptr                pointer unaligned,
B2630                 3 lth                fixed binary (18),
B2631               2 pad                  (4) bit (36);
B2632
B2633
B2634        dcl  request_data_ptr         pointer;
B2635
B2636
B2637     Notes:  This is a replaceable procedure.  See the Programmer's
B2638     Reference Manual for information on the use of replaceable procedures
B2639     within interactive subsystems.  The ssu_request_mgr_$locate_request
B2640     subroutine implements this function by default.
B2641
B2642
Preceding:
A1667     :Entry:  print_blast:  1983-01-19  ssu_$print_blast


A2460     normally perform a non-local goto back to a label in the command's main
A2461     procedure so that it can clean up and return to its caller
Changed by B to:
B3436     normally perform a non-local goto back to a label in the command's
B3437     main procedure so that it can clean up and return to its caller.
B3438
B3439
B3440     :Entry:  token_id:  2025-05-31  ssu_$token_id
B3441
B3442
B3443     Function:  Given a pointer to a token within a token tree returned by
B3444     the ssu_$get_tokens subroutine, this function returns the token's
B3445     identifier locating the token's position within the tree.
B3446
B3447
B3448     Syntax:
B3449     declare ssu_$token_id entry (ptr, ptr, ptr) returns (char(52) var);
```

```
B3450
B3451     token_id = ssu_$token_id (sci_ptr, token_ptr, parent_token_ptr);
B3452
B3453
B3454     Arguments:
B3455     sci_ptr
B3456        is a pointer to the subsystem control structure for this invocation
B3457        returned by either ssu_$create_invocation or
B3458        ssu_$standalone_invocation.  (Input)
B3459     token_ptr
B3460        is a pointer to the token structure whose identifier is to be
B3461        returned.  (Input)
B3462     parent_token_ptr
B3463        is a pointer to a parent token of token_ptr->token.  If this is a
B3464        null pointer, then token_id is a full identifier which locates
B3465        token_ptr->token within the entire token tree.  If it is a non-null
B3466        pointer, then token_id locates token_ptr->token within the sub-tree
B3467        headed by the given parent_token_ptr.
B3468
B3469
B3470     Examples:
B3471     The example below shows a token tree returned by ssu_$get_tokens for
B3472     the string:
B3473
B3474          "a bc (def lmn) ghi[43]"
B3475
B3476     Each token describes a lexeme found by ssu_$get_tokens in this input
B3477     string.
B3478
B3479
B3480          first_token_ptr
B3481            |
B3482            |     token
B3483            |      .nextP    .nextP          .nextP    .nextP
B3484             -->  _   -->  __   -->  _________  -->  ___  -->  ____
B3485                  a       bc       (def lmn)       ghi       [43]
B3486                  -       --       ---------       ---       ----
B3487                                       | .childP                | .childP
B3488                                       |                        |
B3489                                       |        .nextP          -->  __
B3490                                       -->  ___  -->  ___             43
B3491                                            def       lmn            --
B3492                                            ---       ---
B3493
B3494
B3495     ssu_$get_tokens returns a first_token_ptr argument which points to the
B3496     first of 5 top-level token structures.  token.nextP of each token
B3497     points to next sibling token found by ssu_$get_tokens.  If the
B3498     list_rescan_sw argument was set to "1"b, then each token describing a
B3499     list-lexeme (e.g., token.flavor = ParenList) has a token.childP value
B3500     pointing to a subsidiary list of tokens describing lexemes found within
B3501     the list-lexeme string.  The parenthesized list "(def lmn)" holds two
B3502     sub-lexemes: "def" and "lmn".  The bracketed list "[43]" holds one
B3503     sub-lexeme: "43".  A child token describes each of those sub-lexemes.
B3504
B3505
```

```
B3506     ssu_$display_tokens displays information about each token in the token
B3507     tree.  For each token, its token.I value gives an ID nibble which
B3508     locates that token relative to siblings at the same level of the token
B3509     tree; token.I = 1 for first token, ..., token.I = n for the nTH token.
B3510
B3511     Each token with non-null token.childP introduces a new nibble in the
B3512     identifier.  For example, an ID = 3.1 starts with the third TOP-LEVEL
B3513     token, then selects its first child token: the token describing the
B3514     "def" lexeme in the tree shown above.
B3515
B3516
B3517          Tokens by ID
B3518           1: a                                      \
B3519           2: bc                                      \
B3520           3: (def lmn)   | ParenList hasChild         >   TOP-LEVEL TOKENS
B3521           4: ghi                                     /
B3522           5: [43]   | SquareList hasChild Int=43    /
B3523
B3524           3.1: def   | hasParent                    \     CHILD TOKENS of
B3525           3.2: lmn   | hasParent                    /      token ID: 3
B3526
B3527           5.1: 43   | hasParent Int=43              >     CHILD TOKEN of
B3528                                                             token ID: 5


A2471                                                          END HISTORY COMMENTS */
A2472
Changed by B to:
B3538       2) change(2025-06-26,GDixon):
B3539           A) Resolve info seg formatting errors reported by verify_info.
B3540           B) Document the replaceable procedures:  ssu_$invoke_request,
B3541              ssu_$locate_request.
B3542           C) Document new entry points:  ssu_$get_token_area,
B3543              ssu_$get_tokens, ssu_$display_tokens, ssu_$token_id, and
B3544              ssu_$find_token_with_id.
B3545           D) Document child_token added to every token.flavor=CharString
B3546              token.childP to provide quick access to the unquoted version
B3547              of that quoted-char-string (without surrounding double-quote
B3548              chars, and with any escape double-quote pairs in the string
B3549              reduced to a single double-quote char).
B3550
B3551                                                       END HISTORY COMMENTS */

Comparison finished: 13 differences, 1125 lines.
```

The info segment describing the new virtual_ptr (vptr) command/active-function is shown below.

```
2025-12-10  virtual_pointer, vptr

Syntax as a command:  vptr VIRTUAL_PTR {-ctl_args}

Syntax as an active function:  [ vptr VIRTUAL_PTR {-ctl_args} ]

Function:  Converts a VIRTUAL_PTR string to a pointer value by
initiating the referenced segment will a null reference name.  A
VIRTUAL_PTR string is a pathname, reference_name or segment number,
with an optional offset (a number of words or bits) or an
entrypoint_name.  Then vptr displays or returns a character
representation of the virtual pointer which locates the data bit within
the referenced segment as shown below.

   Control_args given       Displayed or Returned Pointer
   ------------------       -------------------------------------
     -ring                    segno|wordno(bit_offset)[ring_number]
     -no_ring -bit            segno|wordno(bit_offset)
     -bit                     segno|wordno(bit_offset)
     -no_bit                  segno|wordno

If neither -bit nor -no_bit is given, a bit_offset component is
displayed or returned only when the given bit_offset value is nonzero.
If neither -ring nor -no_ring is given, a ring_number component is
displayed or returned only when ring_number < ring-of-execution in
which the virtual_pointer command or active function is called.

This normalized representation is useful when creating an abbreviation
or exec_com that examines contents of the location identified by the
virtual pointer.

Upon completion of the abbreviation or exec_com, the virtual_pointer
command may be called a second time with an identical VIRTUAL_PTR
argument and the -terminate control argument.  This terminates any
segment initiated by the earlier call to vptr.

Arguments:
VIRTUAL_PTR
   Character string representation of a pointer.  Format is one of
   those described in the "List of virtual pointer formats" section
   below.

Control arguments (output):
-bit
   Include a decimal bit_offset number in the returned result.
-nonzero_bit, -nzbit
   Include a decimal bit_offset number in the returned result if it is
   nonzero. (default)
-no_bit, -nbit
   Exclude any bit number from the returned result.
```

```
-null
   Return a null pointer value as the word: null
-no_null, -nnull
   Return a null pointer value as a segno and wordno: -1|1
   (default)

-ring
   Always include a ring number in the returned result.
-no_ring, -nring
   Never include a ring number in the returned result.

Control arguments (termination):
   These control arguments may only be given when invoking
   virtual_pointer as a command.
-terminate, -tm
   Terminate the segment referenced by the given VIRTUAL_PTR
   representation in the current ring of the user's process.
   For details, see the: "Notes on terminating the segment" section
   below.
-no_terminate, -ntm
   Do not terminate the segment referenced by the given VIRTUAL_PTR
   representation. (default)

List of virtual pointer components:
segno
   segment number of a segment known to the process.  It is an octal
   number from 0 to 7777 inclusive; or one of the null pointer segment
   numbers (-1 or 77777 or 777777).
ref_name
   reference name for a segment.  It has the format of a PL/1
   identifier up to 32 characters in length.  See "Notes" below.
path
   absolute or relative pathname of a segment or multi-segment file
   (MSF).  It may also use the archive convention (path::component) to
   select one archive component.

entry_pt
   named location within an object segment; or an object MSF
   (multi-segment object file).  It is a PL1 identifier from 1 to 256
   characters in length.  The name designates the location of a word
   somewhere within the object file.  See "Notes" below.

W
   octal word offset from the beginning of the segment.  It may have a
   value from 0 to 777777 inclusive.
B
   decimal bit offset within the selected word of a segment.  It may
   have a value from 0 to 35 inclusive.
```

```
R, r
   effective ring number of a pointer to a segment.  A program
   executing in any ring <= R may reference through that pointer to
   locate a selected data word or bit.  R or r is an octal digit from
   0 to 7 inclusive.  Any ring number must appear after the W word
   number or the entry_pt name of the virtual pointer.  See the section
   "Notes on ring number" for more details.

List of virtual pointer formats:
   The VIRTUAL_PTR argument must follow one of the formats listed
   below.
segno|W(B)[R]
   points to octal word W, decimal bit B within that word, of the
   segment whose octal segment number is segno.  R gives a desired ring
   number to be used in the pointer.
segno|W(B)
   same as segno|W(B)[r] where r is the current ring in which vptr
   was invoked.
segno|W
   same as segno|W(0)[r].
segno|, segno
   same as segno|0(0)[r].

segno|entry_pt[R]
   points to the word located by entry point entry_pt in the object
   file whose octal segment number is segno.  For an object MSF, segno
   must identify component 0 in which all definitions are stored.  If
   entry_pt locates a word in some other component of that object MSF,
   that component's segno will be returned in the resultant
   VIRTUAL_PTR.  R gives a desired ring number to be used in the
   resultant pointer.
segno|entry_pt
   same as segno|entry_pt[r] where r is the current ring in which vptr
   was invoked.

-name segno|...,
-nm segno|...
   treat a segment identifier which looks like a segment number as an
   entryname.  The ... may be any of the OFFSET formats shown earlier.
   Search for a segment with that entryname using linker search paths,
   and make that segment known to the user process.

ref_name$entry_pt[R]
   points to word located by entry point entry_pt in an object file
   whose reference name is ref_name.  R gives the desired ring number
   to be used in the pointer.
ref_name$entry_pt
   same as ref_name$entry_pt[r] where r is the current ring in which
   vptr was invoked.
```

```
ref_name$W(B)[R]
   points to octal word W, decimal bit B of segment or MSF whose
   reference name is ref_name.  If ref_name is a reference name on a
   multisegment file (a reference name on component 0 of the MSF), the
   word and bit offsets are applied within component 0.  R gives the
   desired ring number to be used in the pointer.
ref_name$W(B)
   same as ref_name$W(B)[r] where r is the current ring in which vptr
   was invoked.
ref_name$W
   same as ref_name$W(0)[r].
ref_name$
   same as ref_name$0(0)[r].

-name ref_name$...,
-nm ref_name$...
   treat a segment identifier which looks like a reference name as an
   entryname.  The ... may be any of the OFFSET formats shown earlier.
   Search for a segment with that entryname using linker search paths,
   and make that segment known to the user process.

path|W(B)[R]
   points to octal word W, decimal bit B of segment or MSF identified
   by absolute or relative pathname path.  If the path given identifies
   a multisegment file, the offset given is in component 0 of the MSF.
   path may be an absolute or relative pathname, or it may be an
   archive pathname (path::component) identifying one archive
   component.  R gives the desired ring number to be used in the
   pointer.
path|W(B)
   same as path|W(B)[r] where r is the current ring in which vptr was
   invoked.
path|W
   same as path|W(0)[r].
path|, path
   same as path|0(0)[r].

path|entry_pt[R]
   points to word identified by entry point entry_pt in the object file
   (segment or MSF) identified by path.  R gives the desired ring
   number to be used in the pointer.
path|entry_pt
   same as path|entry_pt[r] where r is the current ring in which vptr
   was invoked.

dir>entry$entry_pt
   points to word identified by entry point entry_pt in the object file
   identified by pathname dir>entry.
<dir>entry$entry_pt
   points to word identified by entry point entry_pt in the object file
   identified by pathname <dir>entry.
```

```
<entry$entry_pt
   points to word identified by entry point entry_pt in the object file
   identified by pathname <entry.

List of input null pointer formats:
   In the VIRTUAL_PTR input, a null pointer is represented by one of
   the virtual pointer representations:
-1|1
-1
77777|1
77777
777777|1
777777
null
null()

List of output null pointer formats:
   In the displayed or returned output, a null pointer is represented
   in one of two formats:
null  (if -null control is given)
-1|1  (default)

Notes:
If VIRTUAL_PTR begins with a pathname, either a relative and an
absolute pathname may be given.  Archive pathnames are supported.

A ref_name or entry_pt must be in the form of a PL1 <identifier>.

 <identifier> ::= <letter>[<letter>|<digit>|_]...

 <letter>     ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|
                  a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

 <digit>:     ::= 0|1|2|3|4|5|6|7|8|9

A ref_name <identifier> may not be longer than 32 characters.  An
entry_pt <identifier> may not be longer than 256 characters.  Thus, a
VIRTUAL_PTR with the format  ref_name$entry_pt  may not be longer than
289 characters.

Notes on terminating the segment:
The segment identified by a pathname is initiated with a null
reference name to make it known to the process address space.  This
assigns a segment number which can be displayed in the virtual_pointer
result.

When operation on that pathname is complete, invoke the vptr command
with that same VIRTUAL_PTR (starting with a pathname) and the
-terminate control to reduce the null reference name counter for the
segment.  If the counter reaches zero and the segment has no real
reference names in the process, it will be removed from the process
address space.

A vptr -terminate operation will not attempt to terminate a null
```

```
pointer or a ring-0 segment.  It will silently fail to terminate an
inner-ring segment.

Notes on ring number:
The vptr command converts a VIRTUAL_PTR to a result:

   - A segment number of the segment containing the data item located
     by the pointer.
   - A word number locating the start of the data item within the
     segment;
   - A bit offset number of the first bit of data within that word;
   - A ring number in which that was initiated in order to construct
     the pointer.

This result is displayed or returned as a character representation:
   segno|W(B)[R]

The R ring number is usually displayed by tools like ring_zero_dump
(rzd) if displaying a pointer in an inner-ring data segment.  Such
tools usually do not display R if it is greater than the current ring
number.

A null pointer value always has a ring number of 0.  This permits a
pointer set to null() to be passed to another ring of execution while
having a binary content that matches the null() built-in function.

The pointer value returned by the cv_ptr_ function must have its ring
number R >= current ring of execution (the number of the protection
ring in which the caller of cv_ptr_ is executing) unless a null pointer
is returned.  If its input VIRTUAL_PTR has R < current ring, the
pointer display or returned has an effective ring set by the hardware
to the ring of execution of the caller of cv_ptr_.
```

The enhanced version of the virtual_pointers.gi.info segment is shown below.

```
2025-10-10  Virtual Pointers

The cv_ptr_ function converts a virtual pointer character string to a
real pointer within the current process.

A virtual pointer character string may contain one or two parts: a
segment identifier; and an optional offset into the segment.  A variety
of formats are accepted.  They are shown in the list below.

The virtual_pointer (vptr) command/active_function converts an input
virtual pointer to the format:
     segno|W(B)
where segno gives the segment number by which the segment is known in
the user's process, W is a word offset within that segment, and B gives
a bit offset from the beginning of that word.

List of virtual pointer components (segment identifier):
   In the "List of virtual pointer formats" section below, each format
   references one or more of the following components.
segno
   segment number of a segment known to the process.  It is an octal
   number from 0 to 7777 inclusive; or one of the null pointer segment
   numbers (-1 or 77777 or 777777).
ref_name
   reference name for a segment.  It has the format of a PL/1
   identifier up to 32 characters in length.  See "Notes" below.
path
   absolute or relative pathname of a segment or multi-segment file
   (MSF).  It may also use the archive convention (path::component) to
   select one component of an archive segment.

List of virtual pointer components (offset):
entry_pt
   named location within an object file.  It is a PL1 identifier from 1
   to 256 characters in length.  It may not contain a dollar-sign ($)
   character.  The name designates the location of a word somewhere
   within the object file.  See "Notes" below.
W
   octal word offset from the beginning of a segment.  It may have a
   value from 0 to 777777 inclusive.
(B)
   decimal bit offset within that word of the segment.  It may have a
   value from 0 to 35 inclusive.
```

```
[R]
   effective ring number of a pointer to a segment.  A program
   executing in any ring <= R may reference through that pointer to
   locate a selected data word or bit.  R is an octal digit from
   0 to 7 inclusive.  Any ring number must appear after any W word
   number or the entry_pt name of the virtual pointer.  See the section
   "Notes on an inner-ring pointer" for more details.

List of virtual pointer formats:
segno|W(B)[R]
   points to octal word W, decimal bit B of segment whose octal
   segment number is segno.  R gives effective ring for the pointer.
segno|W(B)
   same as segno|W(B)[R] where R is the current ring of execution in
   which the virtual pointer is being evaluated.
segno|W
   same as segno|W(0)[R].
segno|, segno
   same as segno|0(0)[R].

segno|entry_pt[R]
   points to word identified by entry point entry_pt in segment whose
   octal segment number is segno.  If segno identifies component 0 of
   an object MSF, the pointer returned may not point within the segment
   identified since the target of a definition in component 0 of an
   object MSF may identify an entrypoint in another component of the
   object MSF.  R gives the effective ring of the pointer.
segno|entry_pt
   same as segno|entry_pt[R] where R is the current ring of execution
   in which the virtual pointer is being evaluated.

-name segno|...,
-nm segno|...
   treats a segment identifier which looks like a segment number as an
   entryname.  The ... may be any of the OFFSET formats shown earlier.
   cv_ptr_ searches for a segment with that entryname using linker
   search paths, makes that segment known in the calling ring of the
   user process, and returns a pointer to that segment.

ref_name$entry_pt[R]
   points to word identified by entry point entry_pt in the file whose
   reference name is ref_name.  R gives the effective ring of the
   pointer.
ref_name$entry_pt
   same as ref_name$entry_pt[R] where R is the current ring of
   execution in which the virtual pointer is being evaluated.

ref_name$W(B)[R]
   points to octal word W, decimal bit B of that word in the segment or
   MSF whose reference name is ref_name.  If ref_name is a reference
   name on a multisegment file (i.e. on component 0 of the MSF), the
   word and bit offsets are applied within component 0.  R gives the
   effective ring of the pointer.
ref_name$W(B)
```

```
   same as ref_name$W(B)[R] where R is the current ring of execution in
   which the virtual pointer is being evaluated.
ref_name$W
   same as ref_name$W(0)[R].
ref_name$
   same as ref_name$0(0)[R].

-name ref_name$...,
-nm ref_name$...
   treats a segment identifier which looks like a reference name as an
   entryname.  The ... may be any of the OFFSET formats shown earlier.
   cv_ptr_ searches for a segment with that entryname using linker
   search paths, makes that segment known in the calling ring of the
   user process, and returns a pointer to that segment.

path|W(B)[R]
   points to octal word W, decimal bit B of the segment or MSF
   identified by absolute or relative pathname path.  If the path given
   identifies a multisegment file, the offset given is in component 0
   of the MSF.  path may be an absolute or relative pathname, or it may
   be an archive pathname (path::component) identifying one archive
   component.  R gives the effective ring of the pointer.
path|W(B)
   same as path|W(B)[R] where R is the current ring of execution in
   which the virtual pointer is being evaluated.

path|W
   same as path|W(0)[R].
path|, path
   same as path|0(0).
path|entry_pt[R]
   points to word identified by entry point entry_pt in the object file
   identified by path.  R gives the effective ring of the pointer.
path|entry_pt
   points to word identified by entry point entry_pt in the object file
   identified by path.

dir>entry$entry_pt
   points to word identified by entry point entry_pt in the object file
   identified by pathname dir>entry.
<dir>entry$entry_pt
   points to word identified by entry point entry_pt in the object file
   identified by pathname <dir>entry.
<entry$entry_pt
   points to word identified by entry point entry_pt in the object file
   identified by pathname <entry.
```

```
List of null pointer formats:
   A virtual pointer can represent a null pointer value using one of
   the following forms.  All of these map to a real pointer which
   compares equally to the PL/1 null() built-in function: -1|1
-1|1
-1
77777|1
77777
777777|1
777777
null, null()

Notes:
When a virtual pointer begins with a path, either a relative or an
absolute pathname may be given.  An archive pathname may be given: a
pathname using :: to separate the archive segment name from a component
name.  For example, if segment my_source.archive residing in the
working directory contains a component test_program.pl1, an archive
pathname referencing that component would be:
    my_archive::test_program.pl1

A ref_name or entry_pt must be in the form of a PL1 <identifier> which
does not contain a dollar-sign ($) character.

 <identifier> ::= <letter>[<letter>|<digit>|_]...

 <letter>     ::= A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|S|T|U|V|W|X|Y|Z|
                  a|b|c|d|e|f|g|h|i|j|k|l|m|n|o|p|q|r|s|t|u|v|w|x|y|z

 <digit>      ::= 0|1|2|3|4|5|6|7|8|9

A ref_name <identifier> may not be longer than 32 characters.  An
entry_pt <identifier> may not be longer than 256 characters.  Thus, a
virtual pointer with the format  ref_name$entry_pt  may not be longer
than 289 characters.

A segno is an octal number ranging from 0 thru 7777; or one of the null
pointer segment numbers:  -1  77777  777777  null  null()
Processes except the Initializer further limit the highest segment
number to 1777.
```

```
Notes on evaluating a segment identifier:
When SEG_DELIMITER is vertical-bar (|) or is missing, then

 -name segno,
 -name segno |   Treat segno as a pathname.  Search for that entryname
                 using linker search paths.

       segno,
       segno |   If segno has characteristics of a segment number,
                 then:
                  A) See if it matches a segment number known in the
                     current ring of the process, or is a hardcore
                     segment known to all rings of every process.
                  B) If not: search for segno as an entryname using
                     linker search paths.

        path,
        path |   Otherwise, treat the segment identifier as a
                 pathname.
                  A) If path has < or > characters, use
                     expand_pathname_$component to evaluate it as an
                     absolute path, relative path or archive (::) path.
                  B) If no < or > chars appear in path, treat it as a
                     single entryname which is searched for using
                     linker search paths.
                 Use object_lib_$initiate, initiate_file_$component
                 or phcs_$initiate to make that segment known to the
                 user process.

When SEG_DELIMITER is dollar-sign ($), then

 -name ref_name $   Treat ref_name as a pathname.  If it has no < or >
                    chars, search for that entryname using linker
                    search paths.  Otherwise initiate it by pathname.

       ref_name $   If ref_name has characteristics of a reference
                    name, then:
                     A) See if it is a reference_name known in the
                        current ring of this process.  This includes
                        searching for a hardcore ref_name which is
                        known to all rings of every process.
                     B) If not found, treat it as a single entryname
                        which is searched for using linker search
                        paths.
```

```
            path $   Otherwise, treat the segment identifier as a
                     pathname.
                      A) If path has < or > characters, use
                         expand_pathname_$component to evaluate it as an
                         absolute path, relative path or archive (::)
                         path.
                      B) If no < or > chars appear in path, treat it as
                         a single entryname which is searched for using
                         linker search paths.
                     Use object_lib_$initiate, initiate_file_$component
                     or phcs_$initiate to make that segment known to the
                     user process.

Notes on an inner-ring pointer:
The cv_ptr_ function converts a virtual pointer to an even-word-aligned
2-word pointer value that includes:
   - segment number of the segment containing the data item located by
     the pointer;
   - effective ring number in which that segment is known;
   - word number locating the start of the data item within the
     segment; and
   - bit offset number of the first bit of data within that word.

All of these components are shown in the representation: segno|W(B)[R].
Any ring number [R] must appear after any W word number or the entry_pt
name of the virtual pointer.

A null pointer value always has an effective ring number of 0.  This
allows a pointer set to null() to have identical content in every ring
of execution.

If you use a tool like the ring_zero_dump (rzd) command to display an
inner-ring pointer, its display may include the effective ring number
if the pointer addresses an inner-ring segment.

     rzd 231 -as stack_header.stack_end_ptr
     000000
       stack_end_ptr = 231|2000[1]   [pd]>stack_1

The rzd command may only reference segments in rings 1 through 3 if the
user has access to call entry points in the >sl1>phcs_ gate.

The cv_ptr_ function returns a pointer value to the ring of execution
in which that function was called.  Therefore, the effective ring
number of that pointer [R] will be >= that current ring number unless a
null pointer is returned.  If cv_ptr_ is invoked with a virtual pointer
which has [R] < current ring, the Multics access control hardware
forces the effective ring in the actual returned pointer to the ring of
execution of the caller of cv_ptr_.
```

Version History

Date	Revision	Author	Comment
2026-05-06	0.1	Gary Dixon	Initial version of this MCR.
2026-05-10	1.0	Gary Dixon	Testing section added to this MCR.
2026-05-17	1.1	Gary Dixon	Fix page format issues within the MCR.