Subject: MCR10152, Exec_com Scripts for analyze_multics

Author: Gary Dixon

Date: March 30, 2025

Multics Ticket: https://multics-trac.swenson.org/ticket/352
https://multics-trac.swenson.org/ticket/353

Introduction

Ticket #352 proposes an interesting enhancement to the analyze_multics (azm) subsystem.

> Complete implementation of exec_com scripts containing azm requests.

- A user can invoke such script by the azm **exec_com (ec)** request to:
  - select data from Multics tables using the azm display -as STRUCTURE.ELEMENT request;
  - use the exec_com &set statement to temporarily store the data;
  - use Multics active functions to provide data conversion and formatting services; and
  - use the exec_com &print statement to display data to the user who invoked the script.

At the same time, it would be useful to correct a few minor problems with azm info segments (which describe azm requests and facilities). Several nits have been noticed but deemed too insignificant to correct. Since the proposed enhancement will involve adding new requests to azm (with info segment changes), these azm nits can be repaired with very little extra effort.

One of these azm problems is caused by a bug in ssu_ code as described in Ticket #353. It would be useful to correct this problem as well.

Multics vs. azm Exec_com Scripts

A Multics exec_com script contains a sequence of Multics command lines to evaluate, plus version 2 exec_com statements to: control the sequence of statements to execute; and manage and display data with &set and &print statements.

An azm exec_com script contains a sequence of azm request lines to evaluate, along with those same version 2 exec_com controls. The standard ssu_ **execute (e)** request/active_request can be used: to run a Multics command; or to evaluate a Multics active function and return its results to the azm script. For example:

```
&set env      &[e "reverse_after &ec_path ."]
```

However, remembering to use **execute** for each Multics active function needed, and to add an extra level of quoting around the command line passed to execute, etc. can add difficulties to writing a script using only the current azm requests.

The simplest solution to this problem is to add a useful set of Multics string handling, logical and mathematical active functions to the list of azm requests.

The following Multics command/af tools are useful for writing exec_com scripts.

```
COMPARISON OPERATIONS
  equal   greater   less
  nequal  ngreater  nless
  and     or        not


MATH OPERATIONS
  calc
  ceil
  decimal, dec
  divide
  floor
  octal, oct
  quotient
  max
  min
  minus
  plus
  round
  times
  trunc


OTHER OPERATIONS
  call
  contents
  index_set, ixs
```

```
STRING OPERATIONS
  after, af
  before, be
  clock
  decat
  format_line, fl
  format_line_nnl, flnnl
  index
  length, ln
  ltrim
  reverse, rv
  reverse_after, rvaf
  reverse_before, rvbe
  reverse_decat, rvdecat
  reverse_index, rvindex
  reverse_search, rvsrh
  reverse_substr, rvsubstr
  reverse_verify, rvverify
  rtrim
  search, srh
  string
  substr
  translate
  verify
```

Table 1: Multics Active Functions for Exec_coms

Add the Features to Subsystem Utilities

Instead of enhancing analyze_multics to fully support exec_coms, it seems like a better idea to add these features to ssu_ so that abbreviation and exec_com support could be added more easily to many different subsystems.

- The Multics command/af tools could be added as a ssu-provided request table that could be added by any subsystem.
- A new info directory could hold links to the info segments describing the above tools, so these could be easily added to the help provided by any subsystem.
- A new info segment could describe how subsystem users could make use of abbreviations and exec_coms in a subsystem; and how subsystem developers could easily add those facilities to their subsystem.

Using Several Request Tables in a Subsystem

Ticket #353 describes an implementation error in how ssu_ records request tables being added to a subsystem. The position given in the ssu_$add_request_table subroutine call:

```
declare ssu_$add_request_table entry (ptr, ptr, fixed bin,
   fixed bin(35));

call ssu_$add_request_table (sci_ptr, request_table_ptr, position,
   code);
```

does not get recorded as an attribute of the in-use request table. Thus, calls to add another request table do not know where to fit the new table into those in-use tables. Does the new table get added before all the existing tables, or after one of the existing tables, or at the end of the list of in-use tables?

A similar problem exists in the ssu_$add_info_dir mechanism:

```
declare ssu_$add_info_dir entry (ptr, char(*), fixed bin,
   fixed bin(35));

call ssu_$add_info_dir (sci_ptr, info_dir, position, code);
```

This position is not recorded as an attribute of the in-use directory. An attempt to add another info directory does not know where to insert a new directory pathname into the list of in-use info directories.

If this MCR proposes to add another ssu-provided request table and associated info directory for use in many subsystems, these problems will become much more significant. They should be repaired now.

Also, the ssu_request_tables_$standard_requests has included an unimplemented request called **list_request_tables (lrqt)** to identify which request tables are currently in-use. This request should be implemented as part of this MCR. And a similar **list_info_dirs (lid)** request is needed as well.

ssu_ Improvements

The repair for Ticket #353 begins with a change in the structures which hold the in-use request tables and info directories.

Change data structures for ssu_ request tables and info dirs to record a position number for each element of the stored arrays.

A. For request tables, change ssu_request_tables_list.incl.pl1: the request_tables_list structure.

B. For info directories, change ssu_info_dirs_list.incl.pl1: the info_dirs_list structure.

The structures stored within ssu_ internal data are like the .tables(*) and .info_dirs(*) arrays in these structures, thus preserving the position setting between calls to add another item to the array.

The revised structure for request tables is shown below. Former declaration of the .table_position space was:

```
       3 pad bit(36) aligned;
```

```
dcl 1 request_tables_list aligned based (rtl_ptr),
      2 header,
        3 version fixed binary,
        3 n_tables fixed binary,
      2 tables (request_tables_list_n_tables refer (request_tables_list.n_tables)),
        3 table_ptr pointer,
        3 flags,
          4 table_valid bit (1) unaligned,
          4 pad bit (35) unaligned,
        3 table_position fixed binary;

dcl  REQUEST_TABLES_LIST_VERSION_2 fixed binary static options (constant) initial (2);
```

While the `request_tables_list.tables` substructure has the same storage footprint after the change, the differing content of the `tables.table_position` element requires use of a new `request_tables_list.version` value: REQUEST_TABLES_LIST_VERSION_2. The current structure uses version 1.

A similar change is needed in the structure for in-use info directories.

```
dcl 1 info_dirs_list aligned based (idl_ptr),
      2 header,
        3 version fixed binary,
        3 n_info_dirs fixed binary,
      2 info_dirs (info_dirs_list_n_info_dirs refer (info_dirs_list.n_info_dirs)),
        3 info_dirname character (168) unaligned,
        3 uid bit (36),
        3 flags,
          4 info_dir_valid bit (1) unaligned,
          4 pad bit (17) unaligned,
          4 info_dir_position fixed bin(17) unaligned;

dcl  INFO_DIRS_LIST_VERSION_2 fixed binary static options (constant) initial (2);
```

In the revised structure for info_dirs_list shown above, the former declaration for the .pad and .info_dir_position elements was:

```
4 pad bit (35) unaligned,
```

Change ssu_ Code for Request Tables and Info Dirs

Change ssu_ subroutines for request tables and info dirs to make full use of their item position data when adding/setting/listing data items.

A. For request tables, change:
   `ssu_$add_request_table  ssu_$(list set)_request_tables`
   and their target entry point in ssu_request_mgr_.pl1.

B. For info dirs, change:
   `ssu_$add_info_dir  ssu_$(list set)_info_dirs`
   and their target entry point in ssu_info_mgr_.pl1.

In the add entry points, use position value (given as one of its input arguments) to add the item in correct location with respect to position of the existing items. If the new position matches position number of an existing item, the new position is incremented by 1 causing the new item to be stored following that existing item. If the position argument = 0, the new item is placed first in the stored list with a stored position = 1. Any existing item at position = 1 has its position number incremented.

In the list entry points, accept a requested structure version number of 1 or 2. If 1 is requested, return the original structure containing a .pad element, but with its item position stored in the .pad value. If version 2 is requested, item position is returned in the .(table info_dir)_position element.

In the set entry points, accept an input structure with version numbers of 1 or 2. If 1 is received, the .(tables info_dirs) array index is used for each .(table info_dir)_position value. If version 2 is received, ensure the .(table info_dir)_position values in the array increase monotonically.

Add ssu_ Requests to List Request Tables and Info Directories

Implement the `ssu_$list_request_tables_request` and `ssu_$list_info_dirs_request` programs to report the list of request tables/info dirs currently in-use by the subsystem. If invoked as a request, both position and item name are displayed. If invoked as an active request, only the item names are returned separated by a SP character in their saved position order.

A. Add ssu_request_mgr_$list_request_tables_request and ssu_info_mgr_$list_info_dirs_request entry points.

B. Add ssu_.alm changes to transfer:

- ssu_$list_info_dirs_request to ssu_info_mgr_$list_info_dirs_request
- ssu_$list_request_tables_request to ssu_request_mgr_$list_request_tables_request

C. Change ssu_request_tables_$standard_requests to add new requests:

list_request_tables (lrqt)

list_info_dirs (lid)

D. Create new info segments describing these two new requests:

- ssu.list_request_tables.info (ssu.lrqt.info)
- ssu.list_info_dirs.info (ssu.lid.info)

Add these new segments to the ssu.info directory (>doc>subsystem).

An example of output from the new requests is shown below.

```
azm:  list_request_tables

 POSITION  REQUEST TABLE NAME
     1   azm_request_table_
 100000   ssu_request_tables_$standard_requests
 101000   ssu_request_tables_$exec_com_toolkit


azm:  list_info_dirs

 POSITION  INFO DIRECTORY
     1   >user_dir_dir>Multics>GDixon>work>azm_2.5
     2   >doc>subsystem>analyze_multics
 100000   >doc>subsystem>ssu_info_dirs>standard_requests
 101000   >doc>subsystem>ssu_info_dirs>exec_com_toolkit
```

Display Requests Defined in a Named Request Table

The new **list_request_tables (lrqt)** request makes it possible for a subsystem user to get a list of in-use request tables for the subsystem. With these names, the user may then want to list all the requests defined by a particular table.

Change the **list_requests (lr)** request to add a `-table TABLE_NAME` control argument that lists all the requests defined by the named table.

Add an Exec_Com_Toolkit

Create a new ssu_ request table which defines the Multics command/active function programs listed in Table 1: Multics Active Functions for Exec_coms (see above). Change the existing ssu_request_tables_.alm segment to add the table: ssu_request_tables_$exec_com_toolkit.

The multics_request macro that defines each command will have flags.dont_list+flags.dont_summarize to prevent the **list_requests (lr)** and **?** summarize-requests from displaying these requests. However,

list_requests -all will include these requests. The new **list_requests -table exec_com_toolkit** request will also list all these requests.

Add a new >doc>ss>ssu_info_dirs>exec_com_toolkit directory containing links to the >doc>info>XXX.info segments describing the Multics commands into the new toolkit.

Change the ssu_info_directories_.cds segment to add an entry point referencing the new info directory. It currently has one entry point which references the directory:

>doc>ss>ssu_info_dirs>standard_requests

```
dcl ssu_info_directories_$standard_requests char(168) external static;
```

Add a similar entry point to reference:

>doc>ss>ssu_info_dirs>exec_com_toolkit

```
dcl ssu_info_directories_$exec_com_toolkit  char(168) external static;
```

Install a new >doc>info>ssu_exec_com_toolkit.gi.info segment describing tools available in the new toolkit, and how to make them available in a new or existing ssu_ subsystem.

Create an installer script: do_ssu_links.ec which will create the new >doc>ss>ssu_info_dirs>exec_com_toolkit directory, and provision it with links to the >doc>info info segments describing each tool in the toolkit.

This exec_com will also place links in >doc>ss>ssu_info_dirs>standard_requests to the new info segments describing list_request_tables and list_info_dirs requests; and to the new >doc>info>ssu_exec_com_toolkit.gi.info segment.

Change bound-ssu_.bind to add/retain the ssu_request_tables_$exec_com_toolkit entry point; and the ssu_info_directories_$exec_com_toolkit entry point.

Change >doc>info>ssu_.info to describe the enhancements to ssu_$add_info_dir, $add_request_table, $list_info_dirs, $list_request_tables, and $set_info_dirs and $set_request_tables.

analyze_multics Improvements

The analyze_multics (azm) subsystem provides capabilities for displaying data from three different types of address space.

- When analyzing a crash dump, the **select_dump (sld)** request selects a crash dump image (called an FDUMP); then the **select_process (slp)** request focuses on a particular process' address space from that dump image. Requests then focus on both kernel data items and process-related data display.
- When analyzing a process directory saved from a user process that terminated abnormally, the azm **select_deadproc (sldp)** request focuses on data preserved by the `copy_deadproc` command. This command copies process directory segments from the terminated process into the >dumps>saved_pdirs directory.
- When analyzing the saved process directory for a user process that has become non-responsive, the azm **select_deadproc (sldp)** request focuses on data preserved by the `copy_liveproc` command. This command copies process directory segments from the non-responsive user process into the >dumps>saved_pdirs directory.

When analyzing one of these saved_pdir image types, the azm user only has access to data from that saved process directory. So azm commands for dumping kernel data segments are meaningless in that limited address space type. To prevent accidental use of azm kernel requests, the azm **select_deadproc (sldp)** request changes from the full set of azm requests (defined by the azm_request_table_) to a more limited set of requests (defined by an azm_pdir_rq_table_). If the azm user later wants to analyze a crash dump image, the **select_dump (sld)** request switches back to the full set of requests in the azm_request_table_.

azm also includes a standard set of requests provided by most ssu_ subsystems. This ssu_request_tables_$standard_requests table provides requests like: **quit (q), ?** or summarize-requests**, list_requests (lr), exec_com (e), help (h), list_help (lh)**, etc. Thus, two different sets of request tables may be used by azm:

For crash dumps:

```
azm:  list_request_tables
POSITION      REQUEST TABLE NAME
       1      azm_request_table_
  100000      standard_requests
```

For saved pdirs:

```
azm:  list_request_tables
POSITION      REQUEST TABLE NAME
       1      azm_pdir_rq_table_
  100000      standard_requests
```

The simplest way to add a useful set of Multics string handling, logical and mathematical active functions to the list of azm requests would be to define them in a third request table. These extra request definitions would not have to be duplicated in both the azm_request_table_ and azm_pdir_rq_table_.

Add the New Exec_Com_Toolkit

Add a new request table containing Multics active functions useful in formatting and displaying data to the list of requests provided by the azm subsystem

With addition of a new request table, the relative positions of the azm request tables will become as shown below.

For crash dumps:

```
azm:  list_request_tables
POSITION  REQUEST TABLE NAME
      1   azm_request_table_
 100000   ssu_request_tables_$standard_requests
 101000   ssu_request_tables_$exec_com_toolkit
```

For saved pdirs:

```
azm:  list_request_tables
POSITION  REQUEST TABLE NAME
      1   azm_pdir_rq_table_
 100000   ssu_request_tables_$standard_requests
 101000   ssu_request_tables_$exec_com_toolkit
```

All azm address space types will use the same info directories. This will include the directory for the exec_com_toolkit request table.

azm: lid

```
POSITION  INFO DIRECTORY
    1   >udd>m>gd>w>azm
    2   >doc>subsystem>analyze_multics
 100000   >doc>subsystem>ssu_info_dirs>standard_requests
 101000   >doc>subsystem>ssu_info_dirs>exec_com_toolkit
```

Modify analyze_multics.pl1

- Raise the version of the analyze_multics subsystem from 2.4 to 2.5 to reflect addition of this new exec_com capability.
- Define an *azmec* search list which the azm **exec_com (ec)** request can use to locate segments with an *.azmec* suffix. This suffix was added to analyze_multics long ago but never documented or used because of missing features in the azm subsystem. Use of a new search list by the **exec_com** request to locate ec scripts referenced by entryname must be added to azm. This is enabled by calling ssu_$set_ec_search_list.

- To implement the -referencing_dir search rule in the *azmec* search paths, the search mechanism is told to look in the directory containing the current analyze_multics program. This is enabled by calling ssu_$set_ec_subsystem_ptr with a pointer to the analyze_multics program.
- Add the new ssu_request_tables_$exec_com_toolkit to the request tables used by azm.
- Add the new ssu_info_dirs_$exec_com_toolkit to the info directories used by azm.
-

New azm SCRIPT.azmec Segments should be Installed in >tools

Any azm exec_com scripts of general use to dump analyzers should be installed in the Multics Libraries in the >tools directory where bound_azm_ resides alongside a variety of accounting and system operator exec_coms (.ec segments). Info segments describing those new exec_coms should be installed in >doc>privileged (where azm and other system tools are documented).

Storing .azmec and .ec segments in the same directory is recommended because they often provide the same service in different address space types. They can therefore use shared code to perform the common service. The `dump_segment (ds)` and `ring_zero_dump (rzd)` commands offer many of the same facilities for data display which are provided in the azm **display (d)** request.

This change proposes to install the first .azmec script to ensure that new azm exec_com facilities work correctly.

- Install a script to display a fixed bin(71) double-word storage location as a Multics date/time string. The script has two entry points named: `timestamp.ec timestamp.azmec`

  When used as a Multics exec_com script, the tool displays a data location specified via segno|offset, path_name|offset or reference_name$entrypoint_name using the `dump_segment` command (or the `ring_zero_dump` command if the user has access to use the phcs_ gate).

```
ec timestamp pds$first_covert_event_time

  000000157315277444735320o  is:   2025-03-16 03:48:26.967760 pst
```

  When used in azm while analyzing a dump, the same tool selects a location to display using the azm **display** request.

```
azm:  sld >old_dumps>030121.0630.0.7
   ERF 030121.0630.0.7 in directory >old_dumps dumped at 03/01/21  0630.3 pst Mon.

   System-ID MR12.7 Version-ID MR12.7
      Proc   6 DBR  16625434 running on cpu a  Backup.SysDaemon.z

azm:  ec timestamp pds$first_covert_event_time

  000000153555716777203344o  is:   2021-02-03 22:22:45.126884 pst
```

  Contents of the full timestamp.(ec azmec) script has been attached to Ticket #352 for ease of wide-screen display of the script with comments.

Add azmec Search List Defaults

Change search_list_defaults_.cds to define default search rules for the azmec search list:

```
-working_dir
-referencing_dir
>tools
```

The change to analyze_multics.pl1 sets the directory containing the analyze_multics program as the directory searched when the -referencing_dir rule is reached in the azmec search paths. The directory order suggested here allows a person running a special version of the azm command to have that azm's containing directory searched for .azmec segments after the user's working directory, but before >tools dir where prior .azmec scripts have been installed. The dir containing the special version of azm could then hold revised versions of .azmec segments which would be selected before the installed scripts.

Rename the azm "search" Request

This change proposal renames the existing azm **search (srh)** request to **find**. The name "search" collides with the name of the PL1 search builtin function. Its equivalent `search (srh)` command/af will be add as an azm request in the new azm_ab_ec_multics_rq_table_.alm as a tool for writing azm exec_coms.

- Program code in azm_requests_2_.pl1 will have its $search entrypoint renamed to $find.
- The search request documentation segment in >doc>ss>azm>search.info will be renamed to find.info, and contents will refer to the find request (rather than the search request).
- The existing azm search request is defined in both request tables now used in azm:

  azm_request_table_.alm (requests for analyzing dumps);

  azm_pdir_req_table_.alm (requests for analyzing saved process directories).

  The **search (srh)** request must be renamed to **find** in both tables.

Other azm Request Table Changes

- Several existing requests in the current request tables must be repositioned so the requests are listed in a more sensible order. These include: **add_request_table (arqt), configuration_deck (cd), segment_name (name), segment_number (number), syserr_log (slog)**
- The **clock** placeholder request was never implemented. Its placeholder will be removed from the above request tables. The `clock command/af` will be added as one of the tools provided by the exec_com_toolkit.
- The unimplemented **list_request_table (lrqt)** placeholder request will be removed, allowing a new ssu_-provided request to implement this functionality. This new request is one of those soon-to-be-added to the ssu_request_tables_$standard_requests table.

- Finally, several Multics command/af requests were added to these request tables by earlier MCRs to accommodate azm abbreviation defining. These tools overlap the list of exec_com building tools added for ease of azm exec_com scripting. These requests more appropriately belong in the new exec_com_toolkit. They include: **decimal (dec), octal (oct), calc, plus, minus, times, divide, and index_set (ixs)**

Change mbuild to Install azm Exec_coms

Change mbuild (mb) to know how to prepare and install azm Exec_com scripts (.azmec segments). This involves changes to mbuild_compare.pl1 and mbuild_history.pl1 to support these scripts; and changes to mbuild_info_.cds to specify where/how to install such scripts.

Change the Multics history_comment (hcom) command to know how to add/update/display history comments in azm Exec_com scripts. It will use the same comment format specified for the exec_com command's XXX.ec segments.

Build Scripts

The mbuild build scripts for the ssu_, mbuild, and azm changes offers an alternate way of presenting the proposed changes and showing the several components involved in changing these subsystems with associated documentation.

Changes to ssu_

```
Description --
 1) Change data structures for ssu_ request tables and info dirs to record a
    position number for each element of the stored arrays.
      A) For request tables, change ssu_request_tables_list.incl.pl1;
         the request_tables_list structure.
      B) For info directories, change ssu_info_dirs_list.incl.pl1;
         the info_dirs_list structure.
    The structures stored within ssu_ internal data are like the .tables(*)
    and .info_dirs(*) arrays in these structures, thus preserving the
    position setting between calls to add another item to the array.

 2) Change ssu_ subroutines for request tables and info dirs to make full use
    of their item position data when adding/setting/listing data items.
      A) For request tables, change:
           ssu_$(add delete)_request_table   ssu_$(list set)_request_tables
         target entry points in ssu_request_mgr_.pl1.
      B) For info dirs, change:
           ssu_$(add delete)_info_dir   ssu_$(list set)_info_dirs
         target entry points in ssu_info_mgr_.pl1.

    In the add entry points, use position value (given as one of its
    input arguments) to add the item in correct location with respect
    to position of the existing items.  If the new position matches position
    number of an existing item, the new position is incremented by 1 causing
    the new item to be stored following that existing item.  If the
    position argument = 0, the new item is placed first in the stored list.

    In the list entry points, accept a requested structure version number of
    1 or 2.  If 1 is requested, return the original structure containing
    a .pad element, but with its item position stored in the .pad value.
    If version 2 is requested, item position is returned in the
    .(table info_dir)_position element.

    In the set entry points, accept an input structure with version numbers
    of 1 or 2.  If 1 is received, the .(tables info_dirs) array index is
    used for each .(table info_dir)_position value.  If version 2 is
    received, ensure the .(table info_dir)_position values increase
    monotonically.

 3) Implement the long-requested ssu_$list_(request_tables info_dirs)_request
    programs to report the list of request tables/info dirs currently in-use
    by the subsystem.  If invoked as a request, both position and item name
    are displayed.  If invoked as an active request, only the item names are
    returned separated by a SP character in their saved position order.
     A) Change ssu_request_mgr_.pl1 and ssu_info_mgr_.pl1 to add a new
        list_(request_tables info_dirs)_request entry point.
     B) Change ssu_.alm to map its
           ssu_$list_(request_tables info_dirs)_request
        entry points onto those new ssu_(request info)_mgr_ entry points of the
        same name.
     C) Change the ssu_request_tables_$standard_requests to define two new
```

```
      requests which invoke the new ssu_$list_(request_tables info_dirs)_request
      entry points:
         list_request_tables (lrqt)
         list_info_dirs (lid)
   D) Create new .info segments describing these two new requests:
         ssu.list_request_tables.info (ssu.lrqt.info)
         ssu.list_info_dirs.info (ssu.lid.info)
      Add these new info segs to the ssu.info directory (>doc>ss).

4) Add a new exec_com_toolkit table of multics command/af requests useful
   in writing subsystem abbreviations and exec_coms.
   A) In ssu_request_tables_.alm: add a new exec_com_toolkit table containing
      the following multics_requests:

         COMPARISON          STRING MANIPULATION
           equal               after, af
           greater             before, be
           less                clock
           nequal              decat
           nless               format_line, fl
           ngreater            format_line_nnl, flnnl
           and                 index
           not                 length, ln
           or                  ltrim
                               reverse, rv
         MATH                  reverse_after, rvaf
           calc                reverse_before, rvbe
           ceil                reverse_decat, rvdecat
           decimal, dec        reverse_index, rvindex
           divide              reverse_search, rvsrh
           floor               reverse_substr, rvsubstr
           max                 reverse_verify, rvverify
           min                 rtrim
           minus               search, srh
           mod                 string
           octal, oct          substr
           plus                translate
           quotient            verify
           round
           times             MISCELLANEOUS
           trunc               call, cl
                               contents
                               index_set, ixs

   B) Add a new >doc>ss>ssu_info_dirs>exec_com_toolkit directory containing
      links to info segments describing the above Multics commands.
   C) Install a new >doc>info>ssu_exec_com_toolkit.gi.info segment listing
      the tools in the new toolkit, and how to make them available to a new
      or existing subsystem.
   D) Create do_ssu_links.ec script to:
       - Create the new directory in (4-B) and provision it with links
         to the existing info segments describing the Multics command/afs
         listed above.
       - Place link in the >doc>ss>ssu_info_dirs>standard_requests pointing
         to the new segments: >doc>ss>ssu.list_request_tables.info
                              >doc>ss>ssu.list_info_dirs.info
         These links will have same name as the new segments but without the
         leading ssu. prefix.

       - Link to the new segment in (3-C above) from the
            >doc>ss>ssu_info_dirs>standard_requests  directory:
```

```
       lk >doc>info>ssu_exec_com_toolkit.gi.info
          >doc>ss>ssu.exec_com_toolkit.gi.info

         - Link to the new segments in (3-D above) from the
              >doc>ss>ssu_info_dirs>standard_requests  directory:
       lk >doc>ss>ssu.(list_request_tables lrqt).info
          >doc>ss>ssu_info_dirs>standard_requests>(list_request_tables lrqt).info
       lk >doc>ss>ssu.(list_info_dirs lid).info
          >doc>ss>ssu_info_dirs>standard_requests>(list_info_dirs lid).info

    E) Add pathname of that new exec_com_toolkit directory to the existing
       ssu_info_directories_.cds data segment.  It already contains pathname
       of directory holding links to info segments describing requests in the
       ssu_request_tables_$standard_requests:
         dcl ssu_info_directories_$standard_requests char(168) external static;
       The new entry point will be referenced as:
         dcl ssu_info_directories_$exec_com_toolkit char(168) external static;

 5) Change bound_ssu_.bind to:
     - add/retain ssu_request_tables_$exec_com_toolkit external entry point.
     - add/retain ssu_info_directories_$exec_com_toolkit external entry point.

 6) Change ssu_.info to update descriptions of entry points:
      $add_info_dir, $add_request_table, $list_info_dirs,
      $list_request_tables, $set_info_dirs, $set_request_tables

 7) Change the list_requests (lr) request in ssu_request_mgr_.info to
    add the -table TABLE_NAME control argument to display all requests
    in the named request table.  Replace ssu.list_requests.info in the
    ssu.info dir (>doc>subsystem).


Changes --


 Bound_obj:    bound_ssu_                        IN: sss  UPDATE;
   bindfile:   bound_ssu_.bind                    REPLACE;
   source:     ssu_.alm                           REPLACE  compiler: alm;
   source:     ssu_info_directories_.cds          REPLACE  compiler: cds;
   source:     ssu_info_mgr_.pl1                  REPLACE  compiler: pl1 -ot;
   source:     ssu_request_mgr_.pl1               REPLACE  compiler: pl1 -ot;
   source:     ssu_request_tables_.alm            REPLACE  compiler: alm;

 Include:      ssu_info_dirs_list.incl.pl1       IN: sss.include  REPLACE;
 Include:      ssu_request_tables_list.incl.pl1  IN: sss.include  REPLACE;

 Info:         ssu.list_info_dirs.info           IN: ssu.info  ADD;
 Info:         ssu.list_request_tables.info      IN: ssu.info  ADD;
                add_name:  ssu.lrqt.info;
 Info:         ssu.list_requests.info            IN: ssu.info  REPLACE;
                add_name:  ssu.lr.info;
 Info:         ssu_.info                         IN: sss.info  REPLACE;
 Info:         ssu_exec_com_toolkit.gi.info      IN: sss.info  ADD;
                add_name:
                 ssu_exec_com_toolkit.info
                 ssu_toolkit.gi.info
                 ssu_toolkit.info;
```

Changes to mbuild and history_comment

```
Description --
 1) Change mbuild and history_comment to support installing analyze_multics (azm)
    exec_com scripts.
     - These scripts have names ending with a suffix:  .azmec
     - They are processed by exec_com_ and support full exec_com &version 2
       syntax and features.
     - Within an azm invocation, they are invoked by the exec_com (ec) request.
       For example:  ec ENTRYNAME ARGUMENTs ...
     - The given ENTRYNAME is searched for using a new set of search paths
       named:  azmec
     - mbuild has been changed to know how to install them like regular .ec
       segments.
        - The compare (cmp) and history_comment (hcom) requests were trained
          to work on segments with an .azmec suffix.
     - The history_comment (hcom) command was changed to add a history comment
       in .azmec segments in the same fashion as it handles .ec segments.
     - Change mbuild_request_table_.alm to define the new list_request_tables
       (lrqt) request.


Changes --


 Bound_obj:    bound_mbuild_                    IN: tools  UPDATE;
   source:     mbuild_compare_.pl1               REPLACE  compiler: pl1 -ot;
   source:     mbuild_history_.pl1               REPLACE  compiler: pl1 -ot;
   source:     mbuild_info_.cds                  REPLACE  compiler: cds;
   source:     mbuild_request_tables_.alm        REPLACE  compiler: alm;

 Bound_obj:    bound_pnotice_                   IN: tools  UPDATE;
   source:     pnotice_language_info_.cds        REPLACE  compiler: cds;
```

For the changes to analyze_multics and first installed .azmec segment

```
Description --
 azm version 2.5:
 1) Delete >doc>ss>azm>analyze_multics.info segment.  It does not belong there
    and was not getting updated when people changed azm.
 2) Edit >doc>priv>analyze_multics.info segment to add new requests to
    its "List of requests" section.  This was an oversight of MCR10129.
 3) Edit various >doc>ss>azm>**.info segs whose format or content needs updating.
     apte.info                    segment_number.info
     display_absolute.info        select_process.info
     events.info                  set.info
     exec_com.info                syserr_log.info
     find.info
     segment_name.info

 4) Modify analyze_multics.pl1 to:
     - raise version from 2.4 to 2.5;
     - define use of an azmec search list to locate exec_com files use by the azm
       subsystem.  Such exec_coms have an .azmec suffix (already defined by azm).
     - set directory for -referencing_dir search rule to be directory containing
       the analyze_multics command.  The exec_com request will search for .azmec
       segments in that directory when it reaches the -referencing_dir search rule
       in the azmec search paths for the current process.
 5) Change azm_request_table_.alm (used when analyzing dumps) to:
    A) Correct order in which requests are displayed by list_requests and ? request.
    B) Rename the existing azm request:
         search (srh)
       to a new request name:
         find
       to permit the PL/I string-related search (srh) command/AF to be added as
       one of these multics_request entries supporting exec_com writing.
       See change (10) which adds a different search request.
 6) Modify azm_pdir_rq_table_.alm (used when analyzing saved process dirs) for
    similar reasons to those given in (5) above.
 7) Replace search.info (in >doc>ss>azm) with find.info: a new version of search.info
    documenting the new name "find" for this request.
 8) Modify azm_requests_2_.pl1 to rename the $search entry point to $find
    to accommodate change (5) above.
 9) Modify apply.info to provide examples of command_line argument that
    can be used with the apply request of azm.

10) Change azm to add support for exec_com scripts containing azm requests.
     - Within an azm invocation, they are invoked by the exec_com (ec) request.
       For example:  ec ENTRYNAME ARGUMENTs ...
     - These scripts have names ending with a suffix:  .azmec
     - The given ENTRYNAME is searched for using a new set of search paths
       named:  azmec
       (See item 11 below.)
     - They are processed by exec_com_ and support full exec_com &version 2
       syntax and features.

    Add the new exec_com_toolkit request table and info directory to the
    azm configuration.

New exec_coms usable in azm, or via Multics ec command:
    These new exec_coms will be installed in the tools library (in which
    analyze_multics resides).
11) Change search_list_defaults_.cds to define default search paths for .azmec
    segments:  -working_dir  -referencing_dir  >tools
     - Change (4) sets the directory containing the analyze_multics program
```

```
       as the directory searched when the -referencing_dir rule is reached
       in the azmec search paths.
     - The directory order suggested above allows a person running a special
       version of the azm command to have that directory searched for .azmec
       segments after the user's working directory, but before >tools dir.
       The dir containing the special azm could then include revised versions
       of .azmec segments which would be selected before those in >tools.
12) timestamp.(azmec ec) will display a fixed bin(71) data variable as
    a Multics clock reading in iso_long_date_time format.

Link update exec_coms for installer:
13) Add do_azm_unlinks.ec which removes incorrect links from >doc>ss>azm
    directory prior to installing the changes below.
14) Add do_azm_links.ec which installer can run to create these new/changed
    links in the azm.info directory (>doc>subsystem>azm).  The update_seg
    tool does not support install/update operations on links.


Changes --


 Bound_obj:    bound_azm_                        IN: tools  UPDATE;
   bindfile:   bound_azm_.bind                    REPLACE;
   source:     analyze_multics.pl1                REPLACE  compiler: pl1 -ot;
   source:     azm_ab_ec_multics_rq_table_.alm    ADD      compiler: alm;
   source:     azm_pdir_rq_table_.alm             REPLACE  compiler: alm;
   source:     azm_request_table_.alm             REPLACE  compiler: alm;
   source:     azm_requests_2_.pl1                REPLACE  compiler: pl1 -ot;

 Unbound_obj:  search_list_defaults_             IN: sss  REPLACE;
   source:     search_list_defaults_.cds          REPLACE  compiler: cds;

 Info:         analyze_multics.info              IN: azm.info  DELETE;
 Info:         search.info                       IN: azm.info  DELETE;
 Info:         virtual_address.info              IN: azm.info  DELETE;
 Info:         analyze_multics.info              IN: priv.info  REPLACE;
                add_name:  azm.info;
 Info:         apply.info                        IN: azm.info  REPLACE;
                add_name:  ap.info;
 Info:         apte.info                         IN: azm.info  REPLACE;
 Info:         azm.overview.gi.info              IN: azm.info  ADD;
                add_name:
                 azm.overview.info
                 overview.info;
 Info:         display_absolute.info             IN: azm.info  REPLACE;
                add_name:  da.info;
 Info:         events.info                       IN: azm.info  REPLACE;
                add_name:  ev.info;
 Info:         exec_com.info                     IN: azm.info  REPLACE;
                add_name:  ec.info;
 Info:         find.info                         IN: azm.info  ADD;
 Info:         segment_name.info                 IN: azm.info  REPLACE;
                add_name:  name.info;
 Info:         segment_number.info               IN: azm.info  REPLACE;
                add_name:  number.info;
 Info:         select_process.info               IN: azm.info  REPLACE;
                add_name:  slp.info;
 Info:         set.info                          IN: azm.info  REPLACE;
```

```
Info:         syserr_log.info                   IN: azm.info  REPLACE;
               add_name:  slog.info;
Info:         timestamp.ec.info                 IN: priv.info  ADD;
               add_name:  timestamp.azmec.info;
Info:         virtual_address.gi.info           IN: azm.info  ADD;
               add_name:
                virtual_address.info
                VIRTUAL-ADDR.info
                address.gi.info
                address.info;

Seg(exec_com): timestamp.ec                     IN: tools.execution  ADD;
```

Installation Order

Because of the inter-relationships of these three subsystems, they must be installed in a particular order. The following “Notes to the Installer” describe this ordering.

```
-- Installing

   NOTES TO THE INSTALLER: ------------------------------------------

   ORDER OF INSTALLATION:
    1) Install stuff in >udd>m>gd>w>ssu_lr directory.  (MCR10152a)
                   This installs the new ssu_ list_request_tables (lrqt)
                   request into ssu_request_tables_$standard_requests,
                   along with new list_request_tables.info segment.

    1a) Run:  ec ecs>do_ssu_links install
                   This installs links between the directory holding
                   ssu_ standard requests and the directory selecting
                   the standard request info segments for reference by
                   other subsystems.

    2) Install stuff in >udd>m>gd>w>mb directory.      (MCR10152b)
                   This tells mbuild how to install .azmec segments.

    3) Run:  ec ecs>do_azm_unlinks install             (MCR10152c)
                   This script removes links to ssu_ info segs which
                   are replaced by new azm-specific info segs.

    4) In a new process:
       Install stuff in azm.mb build script (through running update_seg)
                   This installs most changes to azm, including use of
                   .azmec scripts and tools for use in those exec_coms.

    5) Run:  ec ecs>do_azm_links   install
                   This script adds links to infos for new exec_com tools
                   to >doc>ss>azm directory, so azm help/list_help requests
                   can find them.  It also references the new
                   list_request_tables.info (lrqt.info) segment.
```

Testing

Tests were divided into three parts: (a) for the changes to ssu_; (b) for the changes to mbuild and history_comment; and (c) for the many changes to analyze_multics.

ssu_ Testing

Changes to ssu_ corrected the methods used by the ssu_$add_request_table and $set_request_tables subroutines. These became much easier to test by implementing the planned **list_request_tables (lrqt)** request, which displays the current tables with their relative position value.

Since analyze_multics subsystem uses four separate request tables (three for each of its investigative environments), it offered plenty of checks that the fix works correctly in interactions with ssu_$add_request_table, $delete_request_table, and **list_requests -all** and **list_requests -table exec_com_toolkit**.

mbuild uses two request tables established via ssu_$add_request_tables. forum uses two request tables established by ssu_$set_request_tables. And each of these subsystems includes ssu_requests_tables_$standard_requests to define some of their requests. This permitted use of **list_request_tables** to display the current request tables in each subsystem. I verified that the revised ssu_ code works correctly in each of these cases.

Testing of the exec_com_toolkit was done using analyze_multics exec_com scripts

mbuild and history_comment Testing

Verifying ability of mbuild and history_comment to handle azm exec_com (.azmec) segments was easy to do. Although the new timestamp.ec script is installed using the .ec suffix, it has an added name of timestamp.azmec. I reordered the names on this segment to put timestamp.azmec first in the name list. I then created a small build_directory to install this new segment into the >tools directory. This gave me the opportunity to test mbuild and history_comment operations through the full steps required to install a .azmec segment. Testing through each applicable mbuild request (**scan, analyze, compare, hcom,** and **install**) then ensured all applicable parts of mbuild worked correctly for .azmec segments.

analyze_multics Testing

The azm changes involve several distinct changes.

- Enabling use of the **exec_com (ec)** request to invoke .azmec scripts.
    - Enabling the azmec search list for finding .azmec scripts not identified by pathname.
- Adding the new azm_ab_ec_multics_rq_table_ with new tools to use in azm exec_com scripts.
    - Making existing info segments for the new tools available to the **azm help (h)** request.
- Making sure the renamed find request is still functioning correctly and has documentation.

- Ensuring that azm exec_com scripts are functional in all azm operating environments (for use in dump images; for use in saved pdirs; for use as Multics exec_coms at command level).
    - Ensuring that all the new tools are accessible to an azm exec_com script.

Tests for: Enabling use of the azm exec_com (ec) request

Two different azm exec_com scripts were created. timestamp.azmec was described earlier in this MCR. A more elaborate event_channel_table.azmec script was created to test this functionality as well. This second script uses functions not currently ready to install, so its installation is being delayed. Both exec_coms operate either as a Multics exec_com (ending in .ec) or as an azm exec_com (ending in .azmec). Both exec_com can be used to display the same data from a user process, or from a dump image or saved process directory image. The exec_coms were placed in differing directories (relative to the working directory and relative to the modified version of analyze_multics). The exec_coms were found:

- When using an absolute pathname: ec [home_dir]>timestamp VIRTUAL-ADDR
- When using a relative pathname: ec <timestamp VIRTUAL-ADDR
- When using just a reference name: ec timestamp VIRTUAL-ADDR
    - with the timestamp.azmec segment in each of the first two pathnames given in the azmec search list: -working_dir and -referencing_dir

In addition, proper error messages were displayed when a non-existent exec_com was named in the ec request line. This validates proper setup for: the .azmec suffix; the azmec search list and its default setting; the azm -referencing_dir directory designation; and actual invocation of the named exec_com.

Tests for: New exec_com tools from the exec_com_toolkit

The two test exec_com scripts referenced many of the new tools added by the exec_com_toolkit without any problem. The other new tools were tested by using them in azm request lines typed at subsystem request level (not in an exec_com script) to be sure they function correctly.

Tests for: Renamed find Request

The find request was tested in all three azm service environments and found to work correctly.

Tests for: Exec_coms in all azm environments

The two exec_com scripts were tested while in different processes of a crash dump image, and when perusing data in a saved process directory. They were also tested as operating as a Multics exec_com in a running user process. The scripts were fully functional in each of these operating environments.

Documentation

Documentation changes for the analyze_multics subsystem are summarized below.

1) Delete >doc>ss>azm>analyze_multics.info segment. It does not belong there and was not getting updated when people changed azm.

2) Edit >doc>priv>analyze_multics.info segment to add new requests to its "List of requests" section. This was an oversight of MCR10129.

3) Edit various >doc>ss>azm>**.info segs whose format or content needs updating.

apte.info	segment_name.info
display_absolute.info	segment_number.info
events.info	select_process.info
exec_com.info	set.info
find.info	syserr_log.info

4) Replace search.info (in >doc>ss>azm) with find.info: a new version of search.info documenting the new name "find" for this request.

5) Modify apply.info to provide examples of command_line argument that can be used with the apply request of azm.

analyze_multics uses the standard ssu_ **help (h)** request implementation. If you type “help” without any arguments, the help request displays the following.

```
azm:  help
Type "help overview" for an overview of the azm subsystem.
Type "?" for a list of available requests.
Type "list_requests" for a short description of the requests.
Type "list_help" for a list of topics available to the help request.
Type "help TOPIC" for more information on a given topic.
```

If an overview.info segment is found in the info directories of the subsystem, the output includes the line recommending “help overview”. This proposal adds the following overview.info segment for the azm subsystem.

6) Add new general information segment giving an overview of analyze_multics operations and requests.

```
NAMES:  azm.overview.gi.info
        azm.overview.info
        overview.info
```

```
2025-03-12  analyze_multics, azm

Function:
The analyze_multics command provides three analysis services.

 A) Analyze captured data describing a crashed Multics system.
     - A system that diagnosed a fatal error and intentionally decided
       to capture a dump (saved memory image) for analysis.
     - A system which became unresponsive caused its System
       Administrator or Operator to manually trigger an Execute Fault
       (XF), which caused the system to capture a dump for analysis.

 B) Analyze captured data describing a user process that terminated
    abnormally.  For details on this capture, type:  help copy_deadproc

 C) Analyze captured data describing a user process which stops
    responding (cannot be interrupted to permit direct analysis by the
    user with probe, debug or trace_stack, etc.).  For details on this
    capture, type:  help copy_liveproc

Notes on crash dump requests:
The following requests are provided to assist with analysis of system
crash images (known as “fdumps”).

azm:  ?

   absolute_address, absadr            page_control_check, pcc
   apply, ap                           page_trace, pgt
   apte                                quit, q
   associative_memory, am              replace, rp
   aste                                scus
   configuration_deck, cd              sdw
   core_map_entry, cme                 segment_name, name
   display, d                          segment_number, number
   display_absolute, da                select_deadproc, sldp
   events, ev                          select_dump, sld
   find                                select_process, slp
   history_regs, hregs                 set
   list_dumps, lsd                     slt_entry, slte
   list_processes, lsp                 stack, sk
   machine_conditions, mc              syserr_log, slog

   traffic_control_queue, tcq          exec_com, ec
   value, v                            execute, e
   verify_associative_memory, vfam     execute_string, exs
   why                                 list_help, lh
   add_request_table, arqt             list_requests, lr
   help, h                             substitute_arguments,
   abbrev, ab                             substitute_args, sbag
   do
```

```
Type "list_requests" for a short description of each request.

Type "help TOPIC" giving the name of any request for details about that
request.
Notes on dead process requests:
The following requests are provided to assist with analysis of an
abnormally-terminating (dead) process, or an unresponsive (live)
process.

azm:  select_deadproc NAME

azm:  ?

   apply, ap              segment_number,       answer
   display, d                number             do
   find                   select_deadproc,      exec_com, ec
   history_regs, hregs       sldp               execute, e
   list_dumps, lsd        select_dump, sld      execute_string, exs
   machine_conditions,    set                   if
      mc                  stack, sk             list_help, lh
   page_trace, pgt        value, v              list_requests, lr
   quit, q                add_request_table,    substitute_arguments,
   replace, rp               arqt                  substitute_args,
   sdw                    help, h                  sbag
   segment_name, name     abbrev, ab

Type "list_requests" for a short description of the requests.

Type "help TOPIC" giving the name of any request for details about that
request.

Notes on requests useful in writing exec_com scripts:
The following requests are useful tools for writing azm exec_com scripts
(XXX.azmec segments).

   call, cl          after, af
   contents          before, be
   if                clock
   index_set, ixs    decat
                     format_line, fl
   equal             format_line_nnl, flnnl
   greater           index
   less              length, ln
   nequal            ltrim
   nless             reverse, rv
   ngreater          reverse_after, rvaf
   and               reverse_before, rvbe
   not               reverse_decat, rvdecat
   or                reverse_index, rvindex
```

```
   calc                reverse_search, rvsrh
   ceil                reverse_substr, rvsubstr
   decimal, dec        reverse_verify, rvverify
   divide              rtrim
   floor               search, srh
   max                 string
   min                 substr
   minus               translate
   mod                 verify
   octal, oct
   plus
   quotient
   round
   times
   trunc

Type "list_requests -table exec_com_toolkit" for a
short description of these requests.

Type "help TOPIC" giving the name of any request for details about that
request.
```

---

7) Add a general information info segment describing analyze_multics VIRTUAL-ADDR arguments.

NAMES: virtual_address.gi.info
virtual_address.info
VIRTUAL-ADDR.info
address.gi.info
address.info

```
2025-03-15  Virtual Address Constructs

Accessing data requires some pointer value to define an address space.
The generation of the pointer value is performed by resolving a virtual
address (a VIRTUAL-ADDR argument to an azm request).  A VIRTUAL-ADDR
consists of two parts:  a segment number and a word offset.

List of virtual address types:
   The azm request resolves VIRTUAL-ADDRs from the following types of
   information.
SYMBOL
   a symbolic name for a segment number and an offset that can be
   resolved by data in the definitions_ segment (i.e., sst$ptl can be
   resolved to the correct segment number and offset of the page table
   lock).
SEGMENT_NAME
   a segment name can be resolved in many ways, but it can only provide
   the first part of a virtual address, a segment number.  azm uses 0
```

as the default offset for the pointer (i.e., tc_data is resolved to SEGNO|0).

SEGMENT_NUMBER
a segment number needs no resolution, but a default action is applied for the offset. The default is a 0 offset: SEGNO|0).

SEGMENT_NAME or SEGMENT_NUMBER and offset
the VIRTUAL-ADDR can reference a segment name or segment number, and give an octal offset (i.e., the construct pds|20 is translated to 71|20). The notation must be used without spaces (e.g., 244|0 or sst$cmp).

TEMPORARY_PTR
azm keeps a set of 11 temporary pointers associated with each selected dump (FDUMP) or selected dead process. These pointers may be set manually with the set request (e.g., set sp 230|100). They can then be referenced by that temporary pointer name as a SYMBOL in a VIRTUAL-ADDR expression. If not set, these pointers are null.

Requests like machine_conditions (mc) and why set all these pointers from information stored in the machine condition data structure. Values of the points can be display using the value request.

OFFSET_OPERATOR
The operations +N and -N immediately preceding an octal number or VIRTUAL-ADDR construct can be used to alter the offset of a virtual address. N is an octal character representation of an integer value. No spaces are allowed between the operator and N. For example: sst$ptl +30 are resolved to be the SEGNO for sst_seg with the offset of the ptl location plus 30 octal words. This can also be expressed with no space between the two constructs:
sst$ptl+30

INDIRECTION
A VIRTUAL-ADDR can imply indirection through a word or pair of words stored at the resolved location. If the location points to a valid ITS pair (it begins on an even-word boundary and the word ends with an ITS (indirect-through-segment) modifier, it is treated as an ITS pointer (pair of words) containing a segment number, word offset and bit offset. azm will reference through that pointer. If the location is not a valid ITS pointer, the upper halfword is used as a segment number. That SEGNO|0 is treated as the indirect reference.

The format of an indirect pointer value is:
segno|offset,* segname|offset,* symbol,*
temp_ptr,* temp_ptr|offset,*

```
Examples:  The follow examples show use of indirection couples with
other VIRTUAL-ADDR formats.

   17|230,*     sst|230,*     sst$cmp,*+2

   sp,*         sp|230,
```

---

8) Add documentation for the azm exec_com request.

NAMES: exec_com.info
ec.info

```
     2025-03-16  exec_com, ec

     Syntax:  ec ec_path {ec_args}

     Syntax as an active request:  [ec ec_path {ec_args}]

     Function:  executes a program written in the exec_com language which is
     used to pass request lines to the analyze_multics subsystem and to
     pass input lines to requests which read input.  As an active request,
     the exec_com program specifies a return value of the exec_com request
     by operands in a &return statement.

     Arguments:
     ec_path
        is the pathname of an exec_com program.  A suffix of .azmec is
        assumed if not specified.
     ec_args
        are optional arguments to the exec_com program and are substituted
        for parameter references in the program such as &1.

     Notes on locating an exec_com segment:
     The analyze_multics subsystem uses the azmec search list to find the
     exec_com program.  The search list is used if ec_path does not contain
     a "<" or ">" character.  If the ec_path does contain either a "<" or
     ">", it is assumed to be a relative pathname.  If the process has not
     defined an azmec search list, a default set of search paths will be
     setup in that process containing:
        -working_dir
        -referencing_dir
        >tools

     Notes on writing an exec_com:
     The exec_com language (version 2) is recommended for writing
     azm scripts.  For a description of this language, type:
         help exec_com.gi

     For exec_com language (version 1), type:  help v1ec.gi

     When evaluating a subsystem exec_com program, subsystem active
```

requests are used rather than Multics active functions when evaluating the &[...] construct and the active string in an &if statement.

The subsystem's execute (e) active request may be used to evaluate Multics active functions within an azm exec_com. For example:

```
   &set pdir_path  &[e process_dir]
```

However, the following Multics active functions found useful in writing exec_com scripts have been added to the azm request list. These additions make writing an azm exec_com more like writing an exec_com that runs at Multics command level.

These azm active requests are displayed only by the request:

```
  list_requests -all
```

so they do not add clutter to the normal list_requests output. They are not displayed by the ? request name summary.

Info segments documenting these added requests have been linked into in the azm help library.

```
  COMPARISON OPERATIONS          SELECTOR OPERATIONS
    equal   greater   less         if
    nequal  ngreater  nless        index_set, ixs
    and     or        not

                                 STRING OPERATIONS
  FILE OPERATIONS                  after, af
    call                           before, be
    contents                       clock
                                   decat
                                   format_line, fl
  MATH OPERATIONS                  format_line_nnl, flnnl
    calc                           index
    ceil                           length, ln
    decimal, dec                   ltrim
    divide                         reverse, rv
    floor                          reverse_after, rvaf
    octal, oct                     reverse_before, rvbe
    quotient                       reverse_decat, rvdecat
    max                            reverse_index, rvindex
    min                            reverse_search, rvsrh
    minus                          reverse_substr, rvsubstr
    plus                           reverse_verify, rvverify
    round                          rtrim
    times                          search, srh
    trunc                          string
                                   substr
                                   translate
                                   verify
```

Notes on limitations:

In the present implementation, any errors detected during execution of an exec_com within a subsystem will abort the request line in which the exec_com request was invoked.

9) Add a timestamp.azmec.info segment to be installed to document the calling sequence for the exec_com.

2025-03-13 ec timestamp

Syntax as a command: ec timestamp VIRTUAL_PTR

Function: Formats a pair of words at a given address as a Multics timestamp string (in iso_long_date_time format).

Arguments:
VIRTUAL_PTR
May be a segment number, name or symbolic address (e.g. 72|332, prds|332, or prds$last_recorded_time) followed by an offset or program entry point name. For more detailed information on acceptable pointers type: help virtual_pointers

Examples:
azm: ec timestamp prds$last_recorded_time

000000153615134141021041o is: 2021-02-28 19:23:39.236897 pst

---

10) Documentation describing use of the exec_com_toolkit, and method of adding it to an existing subsystem.

```
>udd>m>gd>w>lr>ssu_exec_com_toolkit.gi.info   (391 lines in info)

2025-03-27  ssu_ exec_com Toolkit

An ssu_ subsystem may be configured to expand abbreviations referenced
in subsystem request lines.  It may also support capturing a set of
subsystem requests to perform a particular service by storing them in a
subsystem exec_com script.  That service can later be performed by
issuing an exec_com request naming the script and supplying arguments.

Notes on exec_com scripting language:
The version 2 exec_com language is recommended for writing subsystem
scripts.  For a description of this language, use the Multics help
command:   .. help exec_com

For exec_com language (version 1), type:   .. help v1ec

A subsystem request line can contain only subsystem requests.  This is
true even for:
  - executable lines in a subsystem exec_com script;
```

- control statements like &if &[...] that include active string.

The subsystem's execute (e) active request can be used to evaluate Multics active functions which are not part of the subsystem's request table. For example, the active string

```
&set pdir_path  &[e process_dir]
```

invokes the Multics [process_dir] active function to assign the pathname of the current process directory.

The ssu_request_tables_$standard_requests definitions include the execute (e) request.

Notes on the exec_com_toolkit:
Beginning-of-line abbreviations and exec_com scripts often require use of comparison functions, string processing programs, and mathematical operations to correctly provide a complex service.

To reference Multics programs in abbreviations and exec_com statements without using the execute (e) request, ssu_ provides a useful toolkit of Multics programs that can be added as subsystem requests.

The exec_com_toolkit request table provides the following operations.

```
   COMPARISON OPERATIONS          SELECTOR OPERATIONS
     equal   greater   less         if
     nequal  ngreater  nless        index_set, ixs
     and     or        not

   MATH OPERATIONS                STRING OPERATIONS
     calc                           after, af
     ceil                           before, be
     decimal, dec                   clock
     divide                         decat
     floor                          format_line, fl
     octal, oct                     format_line_nnl, flnnl
     quotient                       index
     max                            length, ln
     min                            ltrim
     minus                          reverse, rv
     plus                           reverse_after, rvaf
     round                          reverse_before, rvbe
     times                          reverse_decat, rvdecat
     trunc                          reverse_index, rvindex
                                    reverse_search, rvsrh
                                    reverse_substr, rvsubstr
   OTHER OPERATIONS                 reverse_verify, rvverify
     call                           rtrim
     contents                       search, srh
                                    string
```

```
                                    substr
                                    translate
                                    verify

These Multics command/active function programs are not displayed
by the requests:  list_requests  or ?  (request name summary).
Those operations focus on the central requests of the subsystem.
But information about all available requests is displayed by the
request: list_requests -all

Notes on subsystem abbreviations:
Every subsystem which handles abbreviations in request lines should
support the following control arguments in the command that invokes the
subsystem.  For example, the analyze_multics (azm) subsystem info
segment describes the abbrev-related controls as follows.

   help azm -section abbreviations

   Control arguments (abbreviations):
   -abbrev, -ab
      enables abbreviation expansion of request lines.  If given
      without the -profile control argument, then the default profile
      is used:  [home_dir]>[user name].profile

   -no_abbrev, -nab
      does not enable abbreviation expansion of request lines.
      (Default)

   -profile PATH, -pf PATH
      specifies the pathname of the profile to use for abbreviation
      expansion.  The suffix "profile" is added if necessary.  This
      control argument implies -abbrev.

When running a subsystem, the self-identity (.) request will report the
current subsystem version, and whether abbreviations were enabled by
use of the -abbrev or -profile control arguments.  For example:

   azm -profile [hd]>[user name].azm

   azm:  .
   azm 2.5 (abbrev) No dump selected.

If the subsystem is invoked with abbreviations disabled, the
self-identity request does not show the (abbrev) indicator.

   azm:  q
   r 16:06 0.085 13 level 2

   azm -no_abbrev

   azm:  .
```

```
   azm 2.5 No dump selected.

While in the subsystem, use standard abbrev (ab) requests to list or
modify abbreviations in the chosen profile.

   azm:  .profile
   >user_dir_dir>Multics>GDixon>GDixon.azm.profile

   azm:  .l
   b cmex    execute_string "display sst$cmp,*  -as cma{[decimal &1]}"

   azm:  .?
   Abbrev requests:
       ...
   .? <request1>...<requestN>
      describes the function and usage of the given abbrev control
      request(s). If none are given, all abbrev requests are described.

Notes on exec_com support:
Use the list_requests (lr) request to determine whether the current
subsystem includes the exec_com (ec) request.

   azm:  lr exec_com
   exec_com, ec       Execute a file containing request lines with
                      parameter substitution.

Use the exec_com request to determine what suffix the subsystem expects
to find on its exec_com scripts.

   azm:  ec <foobar
   azm (exec_com): Entry not found.
                   Input file:  >udd>m>gd>w>foobar.azmec

Use the list_help (lh) request to determine if any of the subsystem's
exec_coms have been installed for general use.

   azm:  list_help .azmec
   Topics available for azm:

   timestamp.azmec

Use the help (h) subsystem request to display information about using
those scripts.

   azm:  help timestamp.azmec -all

Standard versions of the help (h) and list_help (lh) requests are
defined by the ssu_request_tables_$standard_requests table.  However,
some subsystems provide their own requests under these names.
```

```
Notes on request tables and info directories:
Two standard requests provide information about which request tables
define the requests provided by the subsystem; and which directories
are searched looking for info segments describing those subsystem
requests and exec_com scripts.

   azm:  list_requests list
    ...
   list_info_dirs, lid          Return list of information directories
                                in use.
    ...
   list_request_tables, lrqt    Return list of request tables in use.

   azm:  lrqt

    POSITION  REQUEST TABLE NAME
          1    azm_request_table_
     100000    ssu_request_tables_$standard_requests
     101000    ssu_request_tables_$exec_com_toolkit


   azm:  lid

    POSITION  INFO DIRECTORY
          1    >udd>m>gd>w>azm
          2    >doc>subsystem>analyze_multics
     100000    >doc>subsystem>ssu_info_dirs>standard_requests
     101000    >doc>subsystem>ssu_info_dirs>exec_com_toolkit

The standard list_requests (lr), list_request_tables (lrqt) and
list_info_dirs (lid) requests are defined by the table:
   ssu_request_tables_$standard_requests

To see which requests are supplied by a given request table, type:
   list_requests -table TABLE_NAME

An example using shorter names:
   lr -tb exec_com_toolkit

Notes on adding exec_coms to a subsystem:
The exec_com (ec) request is one of the requests provided by request
definitions in the ssu_request_tables_$standard_requests request table.
But a subsystem must take several steps beyond including the exec_com
request to fully enable exec_com scripts.

Enhancing a subsystem to support use of abbreviations in request lines
requires changes in the command that enters the subsystem.  That
command must do the following after creating the ssu_ invocation.
```

```
1) Accept the following control arguments:
    -abbrev, -ab
    -no_abbrev, -nab
    -profile PATH, -pf PATH
   See "Notes on subsystem abbreviations" above for descriptions of
   these control arguments.

2) Call ssu_$get_default_rp_options to obtain initial options for the
   subsystem request line processor.  This returns the structure
   documented in ssu_rp_options.incl.pl1.

3) Modify the following options using a pointer to the PATH supplied by
   the -profile argument.
       local_rpo.abbrev_info.expand_request_lines = True;
       local_rpo.abbrev_info.default_profile_ptr = profile_ptr;
       local_rpo.abbrev_info.profile_ptr = profile_ptr;

4) Call ssu_$set_request_processor_options to set the new request
   processor option values.

For details, see descriptions of the ssu_$get_default_rp_options and
ssu_$set_request_processor_options subroutines.



Enhancing a subsystem to support exec_com scripts also requires changes
in the command that enters the subsystem.  That command must do the
following after creating the ssu_ invocation.

1) Define the standard ssu_ exec_com request as one of the subsystem's
   requests.  This can be done by adding a request definition to the
   subsystem's own request table.

      request exec_com,
              ssu_requests_$exec_com,
              (ec),
              (Execute a file containing request lines.),
              flags.allow_both

   Or it can be accomplished by adding a standard set of ssu_-provided
   requests to the array of request tables known to the subsystem.

    call ssu_$add_request_table (sci_ptr,
         addr(ssu_request_tables_$standard_requests), 100000, code);

   Once the command which starts your subsystem has added this request
   table, you can see which requests it includes by typing:

      lr -table standard_requests
```

2) Set details for exec_com script segments to be used by the
   subsystem. Using the analyze_multics system as an example, these
   specify:
    - suffix which ends subsystem exec_com entrynames;
    - name of the search list used to locate exec_com scripts
      referenced only by entryname in the exec_com request;
    - directory to be used if the -referencing_dir search rule
      appears in that search list. In the example below, this is
      set to the directory containing the analyze_multics program.

```
call ssu_$set_ec_suffix       (sci_ptr, "azmec");
call ssu_$set_ec_search_list  (sci_ptr, "azmec");
call ssu_$set_ec_subsystem_ptr(sci_ptr, codeptr(analyze_multics));
```

3) Add an info segment describing the subsystem's exec_com request
   to the subsystem's info directories. This can be done by tailoring
   the analyze_multics subsystem description of exec_com to the
   particular details of your subsystem. Edit the azm segment:

```
   >doc>ss>azm>exec_com.info
```

   Or include the info segment describing the standard exec_com request
   in your subsystem's info directory.

```
   link >doc>ss>ssu.exec_com.info SUBSYS_INFO_DIR>exec_com.info
   add_name SUBSYS_INFO_DIR>exec_com.info ec.info
```

   Or if you are using the standard help (h) and list_help (lh)
   requests, the info segment describing the exec_com (ec) request will
   be accessible in the info directory for the standard requests.

```
dcl ssu_info_directories_$standard_requests char(168) external;

call ssu_$add_info_dir (sci_ptr,
     ssu_info_directories_$standard_requests, 100000, code);
```

4) To your subsystem's request list, add tools useful in defining
   abbreviations and in implementing exec_com scripts. A common set of
   script tools is provided by the subsystem utilities. The tools
   listed in the section titled "Notes on the exec_com_toolkit" can be
   added to your subsystem using an additional request table.

```
call ssu_$add_request_table(sci_ptr,
     addr(ssu_request_tables_$exec_com_toolkit),  101000, code);
```

   The tools declared in this request table can be listed using the
   request:

```
   list_requests -table exec_com_toolkit
```

5) Add info segments describing these abbreviation and exec_com programming tools to your subsystem's list of info directories.

   If your subsystem uses the standard help (h) and list_help (lh) requests, links to the info segments describing these tools are defined in a separate info directory provided by ssu_. Add this to your subsystem's info directories list as shown below.

```
 dcl ssu_info_directories_$exec_com_toolkit char(168) external;

 call ssu_$add_info_dir (sci_ptr,
      ssu_info_directories_$exec_com_toolkit, 101000, code);
```

   If your subsystem uses a different help mechanism, change it to include the info segments addressed as links in the directory:

```
   >doc>ss>ssu_info_dirs>exec_com_toolkit
```

   You can either search this directory for topic names given in help or list_help requests, or copy relevant links into your own info directory.

6) In step 2 above, a search list is established for locating your subsystem's exec_coms using only an entryname rather than a pathname.

```
   call ssu_$set_ec_search_list  (sci_ptr, "azmec");
```

   A subsystem can define an initial set of search paths for that search list by including an unbound object segment in the subsystem's execution directory.

   Follow the example for the azmec search list as defined in the data segment:

```
   >ldd>sss>source>search_list_defaults_.cds
```

   to create a XXX_search_list_defaults_.cds for the subsystem. Compile this source using the create_data_segment (cds) tool to generate an object segment named XXX_search_list_defaults_ with an added name based upon your subsystem's search list name (given in the ssu_$set_ec_search_list call shown above.

   By installing this object segment with your subsystem, the search_paths_ subroutine can use linker search paths to locate the search defaults and set default search paths for the search list.

   In our analyze_multics example, the unbound object is installed with several added names:

```
   >sss>search_list_defaults_
     azmec.search
     ...
```

The standard ssu-provided exec_com request calls search_paths_ to locate the exec_com script. search_paths_ looks in the user process for a defined search list under the list name shown above. If one is not found, search_paths_ uses linker search paths to search for the refname SUBSYSTEM_SEARCH_LIST.search (such as the azmec.search name shown above). It then establishes default settings for that search path which are specified in the azmec.search entry point of that object segment.

The print_search_paths (psp) command follows this same process when displaying a search list that has not been defined in the user's process. Continuing our example, here are the defaults for the azmec search list.

```
   psp azmec

   azmec
            -working_dir
            -referencing_dir
            >tools
```

Version History

Date	Revision	Author	Comment
2025-03-21	1.0	Gary Dixon	Initial draft of this document.
2025-03-22	1.1	Gary Dixon	Fix minor typos, add a few details.
2025-03-30	1.2	Gary Dixon	Move most new features into ssu_ where they can be shared in many subsystems.