Subject: MCR10167, Displaying Multics IPC Event Channel Tables

Author: Gary Dixon

Date: 2026-09-12

Multics Ticket: https://multics-trac.swenson.org/ticket/397
https://multics-trac.swenson.org/ticket/398
https://multics-trac.swenson.org/ticket/399
https://multics-trac.swenson.org/ticket/403
https://multics-trac.swenson.org/ticket/360
https://multics-trac.swenson.org/ticket/406

Inter-Process Communication

Multics software includes an Inter-Process Communication (IPC) mechanism that provides a named unidirectional communication channel between two Multics processes. The ipc_ subroutine creates and manages event channels. A process or system module which detects an occurrence of an event uses the hcs_$wakeup subroutine to notify the creating process of that occurrence.

IPC creates an Event Channel Table (ECT) in each ring of execution in which an event channel is needed. The event channel can be treated as a named slot in the ECT. Events on channels created in a given protection ring are visible only to programs running in that same ring.

Each event channel is either an event-wait, fast event-wait, event-call, or asynchronous event-call channel. These four channel varieties offer distinct modes of responding to received events.

When an ***event-wait channel*** receives an event *wake-up,* an event_message record is appended to its wait_channel slot. If the target process called ipc_$block to wait for an event on that event-wait channel, the wake-up causes the pxss scheduler to resume execution of that process. Or the program that created the channel can call ipc_$read_ev_chn to poll the channel to check for an incoming event. Both of these subroutines remove the event_message record from the wait_channel, and return that message in the event_wait_info structure provided by the caller.

A ***fast event-wait channel*** operates like a regular event-wait channel, but receives no event_message record describing each event. One bit, called the *fast_event_bit*, is set to report receiving a wake-up on that channel. Both ipc_$block and ipc_$read_ev_chn check and clear this bit as they report the incoming fast wait event.

The owner of the fast channel should be able to monitor and clear the fast_event_bit directly. But there is currently no ipc_ interface which returns the location of this bit. Ticket 406 requests a change in the ipc_$create_event_channel interface to return the location of this fast_event_bit to the caller when a fast event-wait channel gets created.

When an ***event-call channel*** receives an event *wake-up,* an event_message record is appended to its call_channel slot. Each report of an event invokes a procedure in the creating process's ring of execution to respond to that event. The handling procedure is specified when the event-call channel is created.

The procedure handling an event-call will be invoked if there are no wake-ups posted in any event-wait channel given in the wait-list passed to ipc_$block.

- If no wait-list channel has received a wake-up: ipc_$block checks the event-call channels in channel-priority order looking for a channel with a pending event_message.
    - If one is found:
        - The handling procedure is invoked with a copy of that event_message, and the channel's caller-supplied data_ptr.
        - When the handling procedure returns, the event_message is removed from the call_channel.
        - This loop continues until all event-call messages have been processed, or until one of the event-wait channels in the wait-list receives a wake-up.
    - If no event-call channel has a pending message, ipc_$block actually blocks the process by calling hcs_$fblock to wait for a wake-up on one of the event-wait channels in its wait-list.

The event-call channels created in a process' ring of execution may also be polled by calling the ipc_$run_event_calls subroutine to check event-call channels in priority order for any pending event.

- If an event is found:
    - The handling procedure is invoked with a copy of that event_message, and the channel's caller-supplied data ptr.
    - When the handling procedure returns, the event_message is removed from the call_channel.

An ***asynchronous event-call channel*** operates like a regular event-call channel except that occurrence of the event causes the channel's handling procedure to be invoked immediately. hcs_$wakeup sends a **wkp_** Inter-Process Signal to the process owning the channel.[1]

When the process that created the async event-call channel receives the wkp_ signal:

- A stack frame is pushed onto its stack to signal a wkp_ condition.
- The static condition handler for the wkp_ condition (wkp_signal_handler_) calls ipc_$run_event_calls to process any asynchronous event-call channels that have received an event message.

[1] A wkp_ Inter-Process Signal can only be sent to the initial ring of execution for a process. Therefore asynchronous event-call channels may work like regular event-call channels if they are created in a ring other than the initial ring (or login ring) of the process.

Thus, async event-call channels process incoming event messages without requiring the process to call ipc_$block or to call ipc_$run_event_calls itself.

For each event type, information about the event channel must be made known to each process that is expected to notice the event by sending a wake-up.

- For an event to be noticed by an explicitly cooperating process, information about the channel is typically placed in a known location of a shared segment. This includes the event channel's name, process_id of the creating process, and expected content of the event message (a 2-word datum) that accompanies all event notifications except those for fast event-wait channels.
- For an event to be noticed by a system module, a subroutine call is typically made to convey event channel name and process_id to the appropriate system module.

Event Handling Problems

It is sometimes necessary to display information from the Event Channel Table (ECT) if the Multics process fails to process incoming wake-ups in a timely fashion.

Ticket 397 requests adding a new tool to:

- Describe event channels in use by a given ring of a process.
- Display information about status of each event channel created in that ring including any pending messages.

The goal of displaying an Event Channel Table (ECT) in the user's ring of execution suggests the following approach:

- Write a PL/I program that includes structure declarations for the items in the ECT.
- Add code which locates those data structures in segments of the current ring. It turns out all of this data is kept in segments in the process directory.
- Figure out which elements to display, and output that data.

Such command should be fairly simple to write.

Then someone asked if the tool could display inner-ring ECTs? Could the design be extended to capture data structures from an inner ring, copy them out to the caller's ring, then display that data?

The ring_zero_peek_ subroutine provides that kind of access to inner-ring data if the user has access to >sl1>phcs_ (privileged hard core services gate). So the design was expanded…

- Write a PL/I program that includes structure declarations for the items in the ECT.
- Add code which locates those data structures in segments of the current ring.
    - If user has access to phcs_ gate, locate such data structures in inner-ring segments and copy those data structures into the user's ring for display.
- Figure out which elements to display, and output that data.

Then someone asked if it would be possible to display the ECTs of another process' address space? Sometimes event handling gets so deadlocked that an administrative process stops responding to typed commands (or to outside requests).

Unfortunately, Multics is designed to protect against access to one process's data by another process. But Multics does permit a System Administrator to capture segments in the process directory of a non-responsive process, and copy them to a protected system location (>dumps>saved_pdirs) for analysis by the analyze_multics (azm) tool. azm has an infrastructure setup to examine another process's address space.

Could our tool's design be expanded to operate in the analyze_multics runtime environment as an azm request?

The answer is no. The azm runtime environment is quite different from the Multics runtime environment. Though the tools would both be displaying the same data structures, it would be simpler to write different programs that are specialized to run in each environment. So two tool programs would be needed.

Then someone suggested that the azm request version of the tool could also be extended to display ECTs from the selected process of a Multics crash dump (known as an *fdump*). Luckily, the azm environment is designed to allow requests to access the address space in a saved process directory, or in the selected process of an fdump with only a few changes to support the differing address spaces.

Then someone pointed out that a running Multics process already has a tool that supports display of data structures in inner rings. It is a command called ring_zero_dump (rzd) which is an alternate entry point in the dump_segment (ds) command.

And analyze_multics has a similar request called display (d) which can also display data structures in any ring of a selected process in an fdump; or in any ring of segments in a saved process directory.

Could those two programs be used to obtain the data, and consolidate actual display of that data into a single program?

The answer is yes. Both runtime environments support exec_com scripts that can invoke Multics commands or azm requests, store selected data, and reformat it for display. So the design changed again.

- Write an exec_com that can determine if it is invoked by the Multics exec_com command, or by the azm exec_com request.
- Use this choice of environment to capture ECT data structures as PL/I structure elements, using the capability of:
  ring_zero_dump PTR -as STRUCTURE
  or display PTR -as STRUCTURE
- Use commands available in both Multics and azm environments to format the captured data and display it to the user. The format_line (fl) command/active_function provides all the formatting capabilities of the ioa_ subroutine within an exec_com script.

Both the ring_zero_dump command and the azm display request use the structure display facility of the Multics probe debugging subsystem to: name the elements in a structure, and show their value based upon PL/I data type declared for the element. For example, the Event Channel Table begins with data stored in its ect_header structure. ring_zero_dump would display that structure as shown on the next page.

ring_zero_dump 247|17240 -as ect_header

```
017240
ect_header         @ 247|17240
  ect_areap = 247|17170   [pd]>!BBBKbjkJDDBDCW.area.linker,
  ect_area_size = 1044, count (-1) = 0, count (0) = 37, count (1) = 36,
  count (2) = 1, count (3) through count (5) = 0
  entry_list_ptrs (1)   @ 247|17252
    firstp = 247|17416   [pd]>!BBBKbjkJDDBDCW.area.linker,
    lastp = 247|20370   [pd]>!BBBKbjkJDDBDCW.area.linker
  entry_list_ptrs (2)   @ 247|17256
    firstp = 247|20406   [pd]>!BBBKbjkJDDBDCW.area.linker,
    lastp = 247|20406   [pd]>!BBBKbjkJDDBDCW.area.linker
  entry_list_ptrs (3)   @ 247|17262
    firstp = null, lastp = null
  entry_list_ptrs (4)   @ 247|17266
    firstp = null, lastp = null
  entry_list_ptrs (5)   @ 247|17272
    firstp = null, lastp = null
  meters                @ 247|17276
    total_wakeups = 0, total_wait_wakeups = 0, total_call_wakeups = 0,
    ittes_tossed = 1
  seed = 2
  flags                 @ 247|17303
    mask_call_count = 0
    OFF: call_priority, wakeup_control_points
  ecit_ptr = 247|17314   [pd]>!BBBKbjkJDDBDCW.area.linker, ecit_lth = 64,
  r_offset = 244201, r_factor = -27848046233,
  last_fast_channel_events = "000000000000"b3
```

Unfortunately, this simple display does not give the user a clear picture of contents of the Event Channel Table. To satisfy this enhancement request, an event_channel_table.ec exec_com could display information in the Event Channel Table ECT) for a particular ring of a process.

Figure 1 on the next page shows data that might be displayed for the ECT in the initial (login) ring of a user process. The section of Figure 1 above the Wait Channels label displays the data from the ect_header structure in a more informative fashion.

```
ec ect header+lists,default+msgs

  GDixon.Multics            2026-08-30 09:01:37
 Ring 4   Event Channel Table      at  247|17240

   --------- CHANNELS -----   Counts
    Total Channels               38
     Wait Channels               37
       Fast Chn in-use            1
       Fast Chn unassigned       35
       Regular Wait Chn           1
     Call Channels                 1

   ---- PENDING MESSAGES --   Counts
              Wait Channel         0
              Call Channel         0
               ITT Channel         0

   ---------- METERS ------   Counts
             Total Wakeups         0
              Wait Wakeups         0
              Call Wakeups         0
                ITT Tosses         1
    Control Points Waiting         0

   ---------- FLAGS -------   ----------------
        Event Call Messages    After Wait Msgs
     Control Point Wakeups        Disabled

  ----------- LISTS -------   First Item
             Wait Channels   at  247|17416
             Call Channels   at  247|20406
 Pending Wait Channel Msgs     (none)
 Pending Call Channel Msgs     (none)
 Pending Interprocess Msgs     (none)

  ---------- STORAGE ------   -------------
                  ECT Area   at  247|17170      size:   1044 words
 Event Channel Index Table   at  247|17314      size:     64 packed pointers


 Wait Channels       starting at  247|17416

                              ----- Wait Channel -----    - Control Points -
                                    Fast_Chan   Count  Wake_CPs
  Address                Name            \   ID  Inh Wkp   |      1st Wkp CP   1st Msg --   Next W Chn
 247|17416  "172127336761400000000001"b3  F   1    0   0              (none)       (none)    247|17434
 247|20424  "172174430730440000000002"b3      -    0   0              (none)       (none)       (none)



 Call Channels       starting at  247|20406

                                  ---- Call Channel ----   - Control Point
                                Priority  Async  Inhib  -Count-   Wake_CPs
  Address                Name            \    |    /    INH    WKP   |       1st Msg --  Next C Chn
 247|20406  "172173071747440000000001"b3  1              0      0             (none)       (none)
            procedure:  325|7233  message_facility_$wakeup_processor     data:  247|176132
            control point ID:  "000234000000"b3
```

FIGURE 1: DEFAULT OUTPUT FOR event_channel_table.ec

A similar ring_zero_dump command can display the value of an individual element of a structure:

```
ring_zero_dump 247|17240 -as ect_header.ect_areap

017240
  ect_areap = 247|17170    [pd]>!BBBKbjkLLghHqf.area.linker
```

A new token command/active function parses the dump_segment structure output into tokens, and then matches tokens with selection specs to return a value to an exec_com `&set` statement.

```
&set areaP
&+ &[vptr |||[token -cmd "ds &(ectP) -as ect_header.ect_areap" -dsav] -null]
```

This sets the `&(areaP)` variable to the value `247|17170`.

Ticket_398 requests installing the token command/active function as an enhancement.

It will be difficult for the new tool to interpret content of the 2-word message datum sent with most wake-ups. These message words have no required format. The content is tailored to the needs of the creator for each kind of event.

```
/* Definition of an event message as returned from ring 0 */

dcl 1 event_message_data  aligned based,
    2 channel_id      fixed bin(71),      /* event channel name       */
    2 message         fixed bin 71),      /* message sent with wakeup */
    2 sender          bit(36),            /* process ID of sender     */
    2 origin,
      3 dev_signal    bit(18) unal,       /* "1"b if device signal    */
                                          /* "0"b if user event       */
      3 ring          fixed bin(17) unal; /* ring of sending process  */
```

The tool will provide only an octal representation of the event_message_data.message for each unprocessed message. Providing meters, and showing status of existing event channels would be a good start for such tool. Interpreting pending event message content can be deferred until needed in the future.

```
azm:  ec ect wait+msgs

  Internet.Daemon           2026-08-10 10:21:18
 Ring 3   Event Channel Table      at  247|17300


 Wait Channels       starting at  247|20446

                             ----- Wait Channel -----    - Control Points -
                                    Fast_Chan   Count  Wake_CPs
   Address                Name          \   ID  Inh Wkp   |      1st Wkp CP   1st Msg --   Next W Chn
  247|20700  "615132315675340000000012"b3      -    0   9              (none)       (none)     374|66
     374|66  "615146300625340000000026"b3      -    0  16              (none)    247|20554       (none)

    Address         Channel Name               Message              Sender      Next MsgP
  247|20554   615146300625340000000026  400003000032400000000001  004700240432      (none)
```

Event Problems - in a system dump or hung process

Sometimes a problem in handling events can stymie operations in a critical system process, or even result in a system crash. The proposed new tool for displaying Event Channel Tables should also run as an analyze_multics (azm) request to display the ECTs in multiple rings of the selected process; or even in the saved process directory captured by the copy_deadproc or copy_liveproc commands.

analyze_multics was recently enhanced to support exec_com scripts operating in the address space of a dumped process. And the azm display (d) request shares most of dump_segment's capability to display data from the dump as a PL/I structure.

What's missing is dump_segment's ability to get structure symbol information from a specific object segment compiled with the pl1 -table option using an -in control argument.

Ticket_399 requests adding the -in control_arg to azm's display request, as described below. With that capability, it should be easy for an exec_com script to display ECT data by using dump_segment in a running Multics process; or using azm's display request against a system crash or saved pdir.

```
help dump_segment -scn structure

>doc>info>dump_segment.info   (24 lines follow; 372 lines in info)

2026-06-12  dump_segment, ds

Control arguments (structure display as a command):
   The following control arguments may not be given when dump_segment
   is invoked as an active function.
-as STRUCTURE_NAME
   displays the data as a PL/I structure defined by STRUCTURE_NAME.
   The STRUCTURE_NAME can be a structure defined in one of the system
   include files supported by the analyze_multics display request.
   See >doc>ss>azm>structure_names.info for a list of supported
   include files and structures.  Or the STRUCTURE_NAME can be a PL/I
   structure declared in an object program specified in the -in control
   argument.  See "Notes for structure display" below.

-in VIRTUAL_ENTRY
   identifies an object program compiled with the -table control
   argument which declares the STRUCTURE_NAME given with the -as
   control argument.  If -in is not given, then STRUCTURE_NAME must be
   found in one of the segments listed in the structure search list.
-long, -lg
   displays each element of the structure on a separate line.
-short, -sh
   displays as many structure elements as will fit on a line, and
   groups elements with common data types together. (Default)
```

Other Minor Improvements

Ticket_403 requests adding the ipc_$read_ev_chn request to the >sss>pl1.dcl file. All of the ipc_ entry points are represented by a transfer vector in the ipc_.alm segment. Therefore, there are no symbols describing the argument list for ipc_ entry points.

To deal with this deficiency, the correct declaration for each ipc_ entry point has been added to the >sss>pl1.dcl data segment, except for ipc_$read_ev_chn and ipc_$run_event_calls. Those entry points were overlooked.

Correcting this deficiency will enable the display_entry_point_dcl (depd) command to display correct syntax for the calling sequence of these entry points; and the emacs pl1dcl function to correctly fill-in the declaration for these entry points.

Ticket_360 requests correcting the declaration of the Multics status_code parameter accepted by entry points in the ring0_get_ subroutine. They currently declare this parameter as:

```
dcl  code  fixed bin;
```

Per Multics coding standards, it should be declared as:

```
dcl  code  fixed bin(35);
```

This mis-declaration prevents depd and emacs pl1dcl from working properly for entry points in ring0_get_.

Contents of `>doc>info>event_channels.gi.info` and `>doc>info>ipc_.info` were reviewed for accuracy in preparing this proposal. Minor corrections or additions were made during this review. Those updated info segments will be part of this change proposal.

Proposed Changes

```
Add new exec_coms usable in azm, or via Multics ec command:
This suite of exec_coms will be installed in the tools library
(the directory in which analyze_multics exec_com request searches
for exec_com scripts ending with the .azmec suffix).

     psp azmec
     azmec
               -working_dir
               -referencing_dir
               >tools


 Ticket_397 - Need a tool to display IPC Event Channels
  E-1) New exec_com to display contents of the Event Channel Table
       maintained in each ring of each process that declares/uses an
       Inter-Process Channel (IPC) channel.
         A) Add event_channel_table.(azmec ec) to the tools
            library.  It will have the short names:  ect.(azmec ec)
         B) Add event_channel_table.(azmec ec).info to the
            doc.priv (>doc>privileged) library.  It will have short
            names:  ect.(azmec ec).info
         C) Add a link to >doc>subsystem>azm info directory locating
            the (event_channel_table ect).azmec.info segment in the
            >doc>priv directory.  This link enables the azm list_help
            request to locate the info segment describing the new
            exec_com.

In bound_exec_com_:
 Ticket_398 - token version 1.1
  T-1) Add new token command/AF to bound_exec_com_ in SSS library.
       Retain both token and token_request entry points.  The
       token$token_request interface acts as an ssu_ request which
       can be added to the ssu_ exec_com_toolkit using a "request"
       macro.
  T-2) Add new token.info to sss.info (>doc>info) library to describe the
       token program as a Multics command/AF.  It has a slightly
       different interface than the azm request.
  T-3) Add a new ssu.token.info to ssu.info (>doc>subsystem) library to
       describe token as an ssu_ request in the ssu_ exec_com_toolkit.


In bound_ssu_:
 Ticket_398 - token version 1.1
  S-1) In ssu_request_tables_.alm, add token to ssu_ exec_com_toolkit.
         A) Add token$token_request as a request to the
            ssu_request_tables_.alm exec_com_toolkit request table so it
            becomes usable to azm and other ssu_ subsystems.
         B) Add a link to >doc>ss>ssu.token.info as token.info request
            documentation in the >doc>ss>ssu_info_dirs>exec_com_toolkit
            directory.
```

In bound_azm_:
Ticket 399 - enhance the analyze_multics (azm) display -as request.
D-1) Modify the azm display request to permit -in OBJECT_PATH to be supported when -as STRUCTURE_NAME has been given. This will override the structure search list, instead looking only in the given OBJECT_PATH for symbol information about STRUCTURE_NAME.
D-2) Modify >doc>subsystem>azm>display.info to describe use of the new -in control argument.
D-3) The display request already supports a -long (-lg) control arg. Add and document the opposing -short (-sh) control argument.

In bound_ipc:
Ticket 406 - change ipc_$create_event_channel to return fast_event_bit ptr.
I-1) Modify ipc_create_arg.incl.pl1 to include new ipc_create_wait_event and ipc_create_fast_event structures. Use of the second structure returns a pointer to the fast_event_bit associated with a fast event-wait channel.
I-2) Modify ipc_real_.pl1 to accept these two new structures by returning a ptr to the fast channel status bit in the ipc_create_arg_structure.call_data_ptr element.

In pl1.dcl:
Ticket 403 - Add ipc_$read_ev_chn to pl1.dcl.
P-1) Include declaration of ipc_$read_ev_chn which was omitted for some reason.

In bound_info_rtns_:
Ticket 360 - ring0_get_ declares Multics status code parameter incorrectly.
R-1) Correct declaration of code parameter in ring0_get_.pl1 and repair internal declarations of called subroutines to accept a fixed bin(35) declaration for code parameter. display_entry_point_dcl and emacs pl1dcl function will then return proper calling sequence for all the ring0_get_ entrypoints.
R-2) Correct Syntax lines in ring0_get_.info to ensure all code parameters are declared fixed bin(35).

Changes --

```
Bound_obj:    bound_exec_com_                IN: sss  UPDATE;
  bindfile:   bound_exec_com_.bind            REPLACE;
  source:     token.pl1                       ADD      compiler: pl1 -ot;

Bound_obj:    bound_azm_                     IN: tools  UPDATE;
  source:     azm_requests_1_.pl1             REPLACE  compiler: pl1 -ot;

Bound_obj:    bound_info_rtns_               IN: sss  UPDATE;
  source:     ring0_get_.pl1                  REPLACE  compiler: pl1 -ot;

Bound_obj:    bound_ipc_                     IN: hard  UPDATE;
  source:     ipc_real_.pl1                   REPLACE  compiler: pl1 -ot;

Bound_obj:    bound_ssu_                     IN: sss  UPDATE;
  source:     ssu_request_tables_.alm         REPLACE  compiler: alm;
```

```
 Include:       ipc_create_arg.incl.pl1           IN: sss.include  REPLACE;

 Info:          display.info                      IN: azm.info  REPLACE;
                 add_name:  d.info;
 Info:          event_channel_table.ec.info       IN: priv.info  ADD;
                 add_name:
                  ect.ec.info
                  event_channel_table.azmec.info
                  ect.azmec.info;
 Info:          event_channels.gi.info            IN: sss.info  REPLACE;
                 add_name:  event_channel.gi.info;
 Info:          ipc_.info                         IN: sss.info  REPLACE;
 Info:          ring0_get_.info                   IN: sss.info  REPLACE;
 Info:          ssu.token.info                    IN: ssu.info  ADD;
 Info:          token.info                        IN: sss.info  ADD;

 Seg(azmec):    event_channel_table.azmec         IN: tools.execution  ADD;

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

---

event_channel_table.(azmec ec)

Addition of the tool to display the Event Channel Table data structures is the biggest change proposed by this MCR. The tool is written as an exec_com script.

It can be invoked in a running process with the command: ec ect …

```
ec ect call+msgs

  GDixon.Multics            2026-09-05 12:40:45
 Ring 4   Event Channel Table      at  247|17240


 Call Channels       starting at  247|20406

                                  ---- Call Channel ----   - Control Point
                                Priority  Async  Inhib  -Count-   Wake_CPs
   Address                Name            \    |    /    INH    WKP   |      1st Msg --  Next C Chn
  247|20406  "312376216731440000000001"b3  1              0      1             (none)       (none)
            procedure:  325|7233  message_facility_$wakeup_processor     data:  247|176132
            control point ID:  "000234000000"b3
```

It can be invoked to display tables from a Multics crash dump by:

- Invoking the analyze_multics (azm) command: azm
- Selecting a crash dump:
  azm: select_dump >old_dumps>011925.2154.0.1
  ERF 011925.2154.0.1 in directory >old_dumps dumped at 01/19/25 2254.7 mst Sun.
- Selecting a process within the dump:
  azm: select_process 0
  Process 0, Initializer.SysDaemon.z, DBR 17007650

- Then displaying the Event Channel Table (ECT) for some ring of that process:
  azm: ec ect ring=1,wait+msgs

```
  Initializer.SysDaemon     2025-01-19 10:40:56
Ring 1   Event Channel Table      at  307|11300


Wait Channels       starting at  307|12446

                           ----- Wait Channel -----    - Control Points -
                                 Fast_Chan   Count  Wake_CPs
 Address                Name            \   ID  Inh Wkp   |      1st Wkp CP   1st Msg --   Next W Chn
307|12446  "702430007046140000000021"b3      -    0   0              (none)       (none)    307|12464
307|12464  "702431221511140000000022"b3      -    0   4              (none)       (none)       (none)
```

It can be invoked to display tables from the saved process directory of a process that was not responding. Such process directory can be captured by the system administrator using the copy_deadproc or copy_liveproc commands. Contents of their process directory is moved to the >dumps>saved_pdirs directory and then examined by analyze_multics.

azm: select_deadproc Internet.Daemon.3.pdir

azm: ec ect wait+msgs

```
  Internet.Daemon           2026-08-10 10:21:18
Ring 3   Event Channel Table      at  247|17300


Wait Channels       starting at  247|20446

                           ----- Wait Channel -----    - Control Points -
                                 Fast_Chan   Count  Wake_CPs
  Address               Name            \   ID  Inh Wkp   |      1st Wkp CP   1st Msg --   Next W Chn
247|20446  "615121127532340000000001"b3      -    0   2              (none)       (none)    247|20464
247|20464  "615122326454340000000002"b3      -    0   1              (none)       (none)    247|20572
247|20572  "615125123242340000000005"b3      -    0   0              (none)       (none)    247|20626
247|20626  "615127521107340000000007"b3      -    0   0              (none)       (none)    247|20700
247|20700  "615132315675340000000012"b3      -    0   9              (none)       (none)     374|66
   374|66  "615146300625340000000026"b3      -    0  16              (none)    247|20554       (none)

   Address         Channel Name                Message              Sender      Next MsgP
247|20554   615146300625340000000026  400003000032400000000001  004700240432      (none)
```

event_channel_table.ec.info

Documentation for displaying the ECT of a running process is shown below.

```
help ect.ec.info -a

>udd>m>gd>w>ect>ect.ec.info   (74 lines in info)

2026-08-16  event_channel_table.ec, ect.ec

Syntax as a command:  ec ect {MODE_STRING {CHANNEL_IDs}}

Function:
Display information in the Event Channel Table (ECT) of your process
current ring of execution; or ECT information in a selected
ring of execution.

Arguments:
MODE_STRING
   is a list of Event Channel Table display options.  Separate options
   using a comma (,) character.   Use no SPACE or HT characters within
   the MODE_STRING argument.  Any mode except ring= may be preceded by
   a circumflex (^) character to disable that display.  See the
   "List of modes" section below for details.

CHANNEL_ID
   Show information only for one or more wait or call channels
   matching the given identifier.  Identifiers may be given as a
   decimal number, as octal number with a trailing radix indicator, or
   as an octal bit string.  For example:
        033704315130440000000006
           OR:
        033704315130440000000006o
           OR:
       "033704315130440000000006"b3

List of modes:
ring=N
   Display the Event Channel Table for ring-N, where N is a positive
   integer from 1 to 7.
ring=initial
   Display the Event Channel Table for the initial ring of execution of
   the process.

header, header+lists
   Display ect_header structure of the Event Channel Table (ECT),
   except for list pointers.  With addition of:
     +lists:    displays pointers to threaded lists of messages from
                the ect_header structure.

call, call+msgs
   Display information for all assigned call channels in the ECT for
   the selected ring of execution.  With addition of:
     +msgs:     displays any wakeup messages below each channel.
wait, wait+msgs
   Display information for all assigned, non-fast wait channels in the
   ECT for the selected ring of execution.  With addition of:
     +msgs:     displays any wakeup messages below each channel.
```

```
fast
   Display information for all assigned, fast wait channels in the ECT
   for the selected ring of execution.

struct
   Use dump_segment ADDR -as STRUCTURE_NAME to display data from
   ect_header, wait_channel, call_channel structures, et al.
version, .
   Display version of this exec_com.
default
   Default modes are:
     ring=initial,header,call,wait,fast,^struct,^version
default+msgs
   Default modes with messages are:
     ring=initial,header,call+msgs,wait+msgs,fast,^struct,^version

Access required:
To display an inner-ring Event Channel Table, ect.ec uses the
ring_zero_dump (rzd) command to access the inner-ring data.  To use
that command, the user must have e effective access to the gate:
  >system_library_1>phcs_

Notes:  For information about the structures displayed, review the
comments in the PL/I include file:  >ldd>incl>ect_structures.incl.pl1
```

---

event_channel_table.azmec.info

Documentation for displaying the ECT of a process selected from a crash dump, or from a saved process directory is shown below.

```
azm:  help ect.azmec -all

   (76 lines in info)

2026-08-16  event_channel_table.azmec, ect.azmec

Syntax:  ec ect {MODE_STRING {CHANNEL_IDs}}

Function:
Display information in one of the Event Channel Tables (ECTs) of the
selected process in a selected ERF dump; or from the saved process
directory created by the copy_deadproc or copy_liveproc commands.  By
default, the script displays the ECT for the initial (or login)
ring-of-execution of the selected process.

Arguments:
MODE_STRING
   is a list of Event Channel Table display options.  Separate options
   using a comma (,) character.   Use no SPACE or HT characters within
   the MODE_STRING argument.  Precede a mode with circumflex (^) to
   disable that display.  See the "List of modes" section below for
   details.
```

```
CHANNEL_ID
   Show information only for one or more wait or call channels
   matching the given identifier.  Identifiers may be given as a
   decimal number, as octal number with a trailing radix indicator, or
   as an octal bit string.  For example:
        033704315130440000000006
           OR:
        033704315130440000000006o
           OR:
       "033704315130440000000006"b3

List of modes:
ring=N
   Display the Event Channel Table for ring-N, where N is a positive
   integer from 1 to 7.
ring=initial
   Display the Event Channel Table for the initial ring-of-execution of
   the process.

header, header+lists
   Display ect_header structure of the Event Channel Table (ECT),
   except for list pointers.  With addition of:
     +lists:    displays pointers to threaded lists of messages from
                the ect_header structure.

call, call+msgs
   Display information for all assigned call channels in the ECT for
   the selected ring-of-execution.  With addition of:
     +msgs:     displays any wakeup messages below each channel.
wait, wait+msgs
   Display information for all assigned, non-fast wait channels in the
   ECT for the selected ring-of-execution.  With addition of:
     +msgs:     displays any wakeup messages below each channel.
fast
   Display information for all assigned, fast wait channels in the ECT
   for the selected ring-of-execution.

struct
   Use dump_segment ADDR -as STRUCTURE_NAME to display data from
   ect_header, wait_channel, call_channel structures, et al.
version, .
   Display version of this exec_com.
default
   Default modes are:
     ring=initial,header,call,wait,fast,^struct,^version
default+msgs
   Default modes with messages are:
     ring=initial,header,call+msgs,wait+msgs,fast,^struct,^version

Access required:
When using the analyze_multics ect.azmec script, information from all
rings is available to the azm display request.  The user only requires
access to the crash dump segment; or to saved process directory of the
dead-process or live process, and to segments in that directory.

Notes:  For information about the structures displayed, review the
comments in the PL/I include file:  >ldd>incl>ect_structures.incl.pl1
```

token.pl1

Source for the new token command/AF and its token$token_request ssu_ request and active request won't be shown here. Documentation for the two interfaces (token command and ssu_ token request) is included in the Documentation section of this MCR.

azm_requests_1_.pl1

Source for the azm display request resides in the azm_requests_1_.pl1 segment. Changes needed for the "display ADDR -as STRUCTURE -in OBJECT_PATH" include the new -in control argument and a -short (-sh) control.

display already has a -long (-lg) control argument but is missing a -short (-sh) control arg. -short is the default. When -long is given, each element of the structure is displayed on a separate line of output. Use of -in is modeled after dump_segment which has a -short (-sh) control to reverse meaning of the -long control argument. So -short will be added to the display request.

The changes made in the display request are shown in the compare_ascii output below.

```
compare_ascii >ldd>tools>source>bound_azm_.s.archive::azm_requests_1_.pl1 ==

Inserted in B:
B66         6) change(2026-08-13,GDixon):
B67             A) Change request   display ADDR -as STRUCTURE_NAME  to support
B68                   -in STRUCTURE_OBJ_PATH
B69                   -short
Preceding:
A66                                                          END HISTORY COMMENTS */


A221      dcl  struct_sw bit (1);
Changed by B to:
B225      dcl  struct_obj_path char(168);
B226      dcl (struct_sw, struct_obj_sw) bit (1);


Inserted in B:
B379   dcl  structure_find_$pathname  entry (char (*), char (*), ptr, fixed bin (35));
Preceding:
A374   dcl  structure_find_$search    entry (char (*), pointer, fixed bin (35));


A1026               expand_ptr_sw, long_sw, struct_sw = "0"b;
Changed by B to:
B1032               expand_ptr_sw, long_sw, struct_sw, struct_obj_sw = "0"b;
```

```
Inserted in B:
B1101                   else if arg = "-in" then do;
B1102                      if af_sw then go to DUMP_AF_ARG_ERR;
B1103                      struct_obj_sw = "1"b;
B1104                      call get_next_arg("structure_object_path", argp, argl);
B1105                      struct_obj_path = arg;
B1106                      end;
Preceding:
A1095                   else if arg = "-long" | arg = "-lg" then do;


Inserted in B:
B1111                   else if arg = "-short" | arg = "-sh" then do;
B1112                      if af_sw then go to DUMP_AF_ARG_ERR;
B1113                      long_sw = "0"b;
B1114                      end;
Preceding:
A1099                   else if arg = "-inst" | arg = "-i" | arg = "-instruction" then
do;


A1258                call ssu_$abort_line (sci_ptr, 0, "^a not available in the active
request.", arg);
Changed by B to:
B1274                call ssu_$abort_line (sci_ptr, 0, "^a not available in an active
request.", arg);


A2274                   call structure_find_$search (unsubscripted_name, symbol_ptr,
code);
Changed by B to:
B2290                   if struct_obj_sw then
B2291                        call structure_find_$pathname (struct_obj_path,
unsubscripted_name, symbol_ptr, code);
B2292                   else call structure_find_$search (unsubscripted_name,
symbol_ptr, code);

Comparison finished: 8 differences, 26 lines.
```

Changes to the display.info segment are shown below.

```
compare_ascii >doc>subsystem>analyze_multics>display.info ==

A2        1984-11-19  display, d
Changed by B to:
B2        2026-08-13  display, d


Inserted in B:
B67          The following control arguments may be given only when structured
B68          data is being displayed.
Preceding:
A67       -as STRUCTURE_NAME
```

```
Inserted in B:
B82          Usage as an active request is not allowed.  (See the token request
B83          to work around this limitation.)
B84
Preceding:
A80          The structure reference following -as must be a single string,


A83          and substring matching.  Usage as an active request is not allowed.
A84
A85       -long, -lg
A86          displays each element of the structure on a separate line.  This
A87          control argument is only implemented with -as.
Changed by B to:
B88          and substring matching.
B89
B90       -in VIRTUAL_ENTRY
B91          identifies an object program compiled with the -table control
B92          argument which declares the STRUCTURE_NAME given with the -as
B93          control argument.  If -in is not given, then STRUCTURE_NAME must be
B94          found in one of the segments listed in the structure search list.
B95
B96
B97       -long, -lg
B98          displays each element of the structure on a separate line.
B99       -short, -sh
B100         displays as many structure elements as will fit on a line, and
B101         groups elements with common data types together. (Default)


Inserted in B:
B237        2) change(2026-08-13,GDixon):
B238            A) Document support for the -in and -short control arguments
B239               supporting the display ADDR -as STRUCTURE_NAME control.
Preceding:
A223                                                        END HISTORY COMMENTS */

Comparison finished: 5 differences, 29 lines.
```

---

ring0_get_.pl1

Changes made to correct declaration of the code variable in the ring0_get_ subroutine entry points are shown below.

```
compare_ascii >ldd>sss>source>bound_info_rtns_.s.archive::ring0_get_.pl1 ==

A11       ring0_get_: proc;
A12
Deleted by B, preceding:
B11       /* "Adjusted" by Bernard Greenberg, for hc def seg 07/22/76 */
```

```
Inserted in B:
B13       /****^  HISTORY COMMENTS:
B14         1) change(2026-08-31,GDixon):
B15            Adjust declaration of the code parameter used in all entry points to be
B16            declared as fixed bin(35) rather than fixed bin.  This will permit
B17            display_entry_point_dcl and emacs pl1dcl function to provide a correct
B18            parameter declaration for these entry points.
B19                                                        END HISTORY COMMENTS */
B20
B21       ring0_get_: proc;
B22
Preceding:
A15       dcl (sltp1, names_ptr1, defs_ptr1) ptr static init (null),


A19            hcs_$initiate ext entry (char (*), char (*), char (*), fixed bin, fixed bin, ptr,
fixed bin),
A20            get_definition_ entry (ptr, char (*), char (*), ptr, fixed bin);
Changed by B to:
B27            hcs_$initiate entry (char(*), char(*), char(*), fixed bin(1), fixed bin(2), ptr,
fixed bin(35)),
B28            get_definition_ entry (ptr, char(*), char(*), ptr, fixed bin(35));


A33            code fixed bin,
Changed by B to:
B41            code fixed bin (35),

Comparison finished: 4 differences, 18 lines.
```

---

ipc_real_.pl1

Changes were made in the ipc_$create_event_channel subroutine interface to permit creation of a fast event-wait channel to return a pointer to the fast_event_bit: an external static bit that is turned on when a wake-up is received on that fast channel. These changes began with addition of two new structures in the ipc_create_arg.incl.pl1 include segment.

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

A1        /*----------BEGIN ipc_create_arg.incl.pl1-----------------------------------*/
Changed by B to:
B1        /*  START OF:       ipc_create_arg.incl.pl1   *  *  *  *  *  *  *  *  *  *  *  *  * */


A9          1) change(86-08-12,Kissel), approve(86-08-12,MCR7479),
A10            audit(86-10-08,Fawcett), install(86-11-03,MR12.0-1206):
A11            New include file added to support async event channels.
Changed by B to:
B9          1) change(1986-08-12,Kissel), approve(1986-08-12,MCR7479),
B10            audit(1986-10-08,Fawcett), install(1986-11-03,MR12.0-1206):
B11            New include file added to support async event channels.
B12         2) change(2026-08-30,GDixon):
B13             A) Add separate ipc_create_wait_event and ipc_create_fast_event
B14                structures to simplify creating those event wait channel types.
B15             B) Add missing pad1 element to the ipc_create_arg_structure.
```

```
A16       dcl  ipc_create_arg_structure_ptr    ptr;
A17       dcl  ipc_create_arg_structure_v1     char (8) internal static options (constant) init
("ipcarg01");
Changed by B to:
B20       dcl  1 ipc_create_wait_event         aligned automatic,
B21              2 version                     char (8) unaligned
init(ipc_create_arg_structure_v1),
B22              2 channel_type                fixed bin
init(WAIT_EVENT_CHANNEL_TYPE),
B23              2 pad1                        fixed bin,
B24              2 pad2                        variable entry (ptr),
B25              2 pad3                        ptr,
B26              2 pad4                        fixed bin,
B27              2 pad5                        fixed bin;
B28
B29       dcl  1 ipc_create_fast_event         aligned automatic,
B30              2 version                     char (8) unaligned
init(ipc_create_arg_structure_v1),
B31              2 channel_type                fixed bin
init(FAST_EVENT_CHANNEL_TYPE),
B32              2 pad1                        fixed bin,
B33              2 pad2                        variable entry (ptr),
B34              2 event_bit_ptr               ptr                    init(null()),
B35                                                                    /* For fast event wait
channels               */
B36                                                                    /*  -- output ptr which
locates fast_event_bit */
B37              2 pad4                        fixed bin,
B38              2 pad5                        fixed bin,
B39
B40            fast_event_bit                  bit(1) unaligned based
(ipc_create_fast_event.event_bit_ptr);
B41


A20              2 version                     char (8) unaligned,    /* From above. */
A21              2 channel_type                fixed bin,             /* See constants below. */
Changed by B to:
B44              2 version                     char (8) unaligned
init(ipc_create_arg_structure_v1),
B45              2 channel_type                fixed bin,             /* See constants below. */
B46              2 pad1                        fixed bin,


A24              2 call_priority               fixed bin (17);        /* For event call channels
-- who's first? */
Changed by B to:
B49              2 call_priority               fixed bin (17),        /* For event call channels
-- who's first? */
B50              2 pad5                        fixed bin,
B51
B52           ipc_create_arg_structure_ptr     ptr;
B53


A35       /*----------END ipc_create_arg.incl.pl1--------------------------------------*/
Changed by B to:
B64       dcl  ipc_create_arg_structure_v1     char (8) internal static options (constant) init
("ipcarg01");
B65
B66       /*  END OF:         ipc_create_arg.incl.pl1      *  *  *  *  *  *  *  *  *  *  *  * */

Comparison finished: 6 differences, 51 lines.
```

Code changes in ipc_real_.pl1 to implement use of the new structures: mainly deal with returning the pointer to the new fast_event_bit to the caller of ipc_$create_event_channel. These changes are shown in the compare_ascii output below. The new structures have identical elements to those in the ipc_create_arg_structure referenced by the code. So the code references elements from the new structures using the original structure as an overlay for those values.

```
compare_ascii >ldd>hard>source>bound_ipc_.s.archive::ipc_real_.pl1 ==

Inserted in B:
B86         5) change(2026-08-30,GDixon):
B87             A) Add get_fast_channel_ptr internal function which uses the algorithm
B88                of read_fast_ev_chn to locate the channel_index within
B89                ipc_data_$fast_channel_events from the encoded name of fast
B90                event-wait channel, and returns a pointer to the actual bit in
which
B91                events are reported for the named fast event channel.
B92             B) In ipc_real_$create_event_channel, code which creates a fast
B93                event-wait channel now returns to the caller the value returned by
B94                get_fast_channel_ptr in the ipc_create_arg_structure.call_data_ptr
B95                element.
Preceding:
A86                                                     END HISTORY COMMENTS */


A292                if ipc_create_arg_structure.channel_type = FAST_EVENT_CHANNEL_TYPE
A293                then do;
A294                          call hcs_$assign_channel (P_event_channel_name, P_code);
A295                     end;
Changed by B to:
B302                if ipc_create_arg_structure.channel_type = FAST_EVENT_CHANNEL_TYPE
B303                then do;
B304                          call hcs_$assign_channel (P_event_channel_name, P_code);
B305                          ipc_create_arg_structure.call_data_ptr =
get_fast_channel_ptr(P_event_channel_name);
B306                     end;


Inserted in B:
B1309
B1310
B1311     /* Get pointer to assigned fast channel event bit  using channel_index
encoded in channel_name. */
B1312
B1313     get_fast_channel_ptr:
B1314          proc (P_channel_name) returns(ptr);
B1315
B1316     dcl       P_channel_name      fixed bin (71) parameter;
B1317
B1318     dcl       channel_bit_ptr     ptr;
B1319     dcl       channel_index       fixed bin;
B1320
B1321     dcl      (addr, addbitno, null, unspec)
B1322                                   builtin;
B1323
B1324     dcl       1 ev_chn_name       aligned like event_channel_name automatic;
B1325
B1326               if  P_channel_name = 0
B1327               then return (null());
B1328
B1329               unspec (ev_chn_name) = unspec (P_channel_name);
B1330               if ev_chn_name.type = REGULAR_CHANNEL_TYPE
B1331               then return (null());
```

```
B1332
B1333               channel_index = ev_chn_name.unique_id;
B1334               if  channel_index < 1 | channel_index >
length(ipc_data_$fast_channel_events)
B1335               then return (null());
B1336
B1337               channel_bit_ptr = addr(ipc_data_$fast_channel_events);
B1338               if  channel_index > 1   then
B1339                    channel_bit_ptr = addbitno(channel_bit_ptr, channel_index-1);
B1340               return (channel_bit_ptr);
B1341
B1342               end get_fast_channel_ptr;
Preceding:
A1298     %page;

Comparison finished: 3 differences, 53 lines.
```

Documentation for use of the new structures is part of the ipc_.info segment. Changes to that info segment affect other entry points besides ipc_$create_event_channel. So those changes are shown in the Documentation section of this MCR.

---

ssu_request_tables_.alm

Changes to ssu_ add the new token request to the tools in ssu_'s exec_com_toolkit request table. That enables ssu_-based subsystems to make use of token as a request available to exec_com's supported by that subsystem, or as an additional request in the subsystem itself.

```
----------  ssu_request_tables_.alm

compare_ascii >ldd>sss>source>bound_ssu_.s.archive::ssu_request_tables_.alm ==

Inserted in B:
B45       "  4) change(2026-08-06,GDixon):
B46       "      Add new request to the exec_com_toolkit request list.
B47       "       token
B48       "          convert input string to tokens and select tokens to display or
B49       "          return.
Preceding:
A45       "                                                    END HISTORY COMMENTS


Inserted in B:
B474                request   token,token$token_request,(),
B475                          (Extracts data items from an input string.),
B476                          flags.dont_summarize+flags.dont_list+flags.allow_both
B477
Preceding:
A469                multics_request translate,(),

Comparison finished: 2 differences, 9 lines.
```

---

pl1.dcl

The ipc_$read_ev_chn declaration sequence is being added to pl1.dcl to overcome use a the ipc_.alm transfer vector for managing entry points to the ipc_ subroutine.

```
----------  pl1.dcl

compare_ascii >sss>pl1.dcl ==

A328      ipc_$reconnect               entry (fixed bin(71), fixed bin(35))
Changed by B to:
B328      ipc_$read_ev_chn             entry (fixed bin(71), fixed bin, ptr, fixed bin(35))
B329      ipc_$reconnect               entry (fixed bin(71), fixed bin(35))
B330      ipc_$run_event_calls         entry (fixed bin, fixed bin(35))

Comparison finished: 1 difference, 4 lines.
```

Testing

This section of the MCR describes how the proposed changes were tested for correct implementation, suitable performance, etc.

event_channel_table.ec

Testing the new event_channel_table.(ec azmec) script involved check that each of its documented MODE_STRING arguments is properly accepted and causes the expected action; and any other string appearing in MODE_STRING is diagnosed as an error.

Testing verified that the ect_header structure elements are properly displayed by the header and header+lists modes; that the wait_channel structure elements are properly displayed by the fast, wait, and wait+msgs modes; that the call_channel structure elements are properly displayed by the call and call+msgs modes; and that the event_message structure elements are properly displayed when messages are present and requested by wait+msgs and call+msgs modes and default+msgs modes.

The ability to display actual contents of these event channel structures was tested by use of the struct mode. This mode uses the display LOC -as STRUCTURE command to display name and value of each structure element before the actual structure is displayed by the script.

The ability to select particular wait_channel or call_channel structures was tested by giving one or more CHANNEL_ID arguments following the MODE_STRING.

Finally, the script was tested in each of its supported operating environments: in a running user process; in an analyze_multics system crash for a selected process; and in a saved process directory for a hung or terminating process (as copied by the copy_liveproc and copy_deadproc commands).

Many of these features were shown in earlier sections of this MCR.

token.pl1

The new token program is the largest change in this proposal, and perhaps the most complex in terms of functions performed, interactions with inputs, and display of data. The major function of this program is to identify a few words of output in the data displayed by another program; and then to display the identified string when invoked as a command, or to return selected words from the matched data as an active string.

The ring_zero_dump program can select and display data stored as PL/I structures, including data stored in rings of execution other that of the user. Furthermore, the azm display request offers similar capabilities. However, both of these programs provide structure display only when invoked as a command/request. They reject structure display when invoked as an active_function or active_request. Both of these programs use the probe debugging command's knowledge of PL/I data structures and symbol table descriptors to display data as a PL/I structure. But these probe requests were not designed to operate as an active function. Therefore, objectives of the token program had to be extended.

The token program design objectives include the following:

A. Operate as a Multics command and active function; or as an ssu_ request and active request.

B. Accept an input string argument; or capture output of another command or request as an input string (performing the same function as file_output and revert_output commands).

C. Accept a set of matching specifications which select one or more adjacent tokens from within the set of parsed input tokens.

D. Parse the input string or captured output into meaningful tokens: PL/I <identifier>, <entry_point_name>, operator symbol, punctuation, etc. Parse each matching specification into the same variety of meaningful tokens.

E. Apply specifications to the parsed input, selecting a matching set of adjacent input tokens.

F. Display the matched tokens as output of the token command/request; or select <value> tokens from the matched set of tokens to return as the active string from the token active-function or active-request.

The ssu_ subsystem provides a suitable infrastructure for implementing a Multics command and/or active function; or an ssu_ request and/or active request. So **objective A** was met using ssu_ as the underlying infrastructure. ssu_ functions are well-tested and simple to use correctly. So little testing was required beyond basic operation in all four of those operating modes. The event_channel_table.(ec azmec) script makes use of token's active-function and active-request operating modes. But the command and request modes were tested while the ect.ec script was being implemented to ensure each use of token returned the proper element value. Successful outputs from that script gave proof that all four environments were correctly implemented.

To use these two input sources require **objective B**: capturing command/request output in a file, and processing that data as input to the token command.

To provide any hope for ease-of-use, the token program itself had to capture output from an arbitrary command. Testing this data capture functionality in all four operating modes was fairly straightforward. Making sure of correct recovery from token program interrupt (by a quit signal, or program fault, etc) was ensured by ssu_ mechanisms; as were release of temporary segments and other resources when the token program ended. Proper achievement of **objective B** was therefore fairly easy to test.

The main purpose of token is to extract meaningful tokens from output which displays a PL/I structure, or an individual structure element, by name and value. This can be done using six matching rules if each element is displayed individually. probe has several mechanisms for compressing data if: two or more adjacent array elements share the same value; or if bit(1) values share the same "0"b or "1"b value. By selecting individual elements for display, these compressing mechanisms can be avoided.

Objective C was met by examining the method used by probe to display: a scalar value; a scalar pointer value which was followed by an entrypoint name describing the location pointed to; an element's name and array index followed by its value; a pointer element's name and array index followed by its locator value and description of the location pointer to; a one-bit element name and bit value which was set to ON ("1"b) or OFF ("0"b).

All six of these display formats are produced when displaying structures contained in an Event Channel Table.

The six specifications needed to parse structure display output strings are in BNF-like format:

```
-match  "<element_name> <array_index> = <value> <value: if <entrypoint_name>>"
-match  "<element_name> <array_index> = <value>"
-match  "<element_name>               = <value> <value: if <entrypoint_name>>"
-match  "<element_name>               = <value>"
-match  "ON  : <value: if <element_name> returns "1"b>"
-match  "OFF : <value: if <element_name> returns "0"b>"
```

The first specification looks for an array element whose value is followed by string matching the rules of a PL/I <entrypoint_name>. For example:

```
ptr (2) = 325|7233   message_facility_$wakeup_processor
```

The second specification looks for an array element whose value is not followed by an <entrypoint_name>. For example:

```
count (2) = 1
```

The third and fourth specifications look for a scalar element whose value is either followed by a locator reference <entrypoint_name> or one not followed by such name.

```
ptr = 325|7233   message_facility_$wakeup_processor

count = 1
```

The fifth and sixth specifications look for a bit (1) element whose value is ON or OFF as displayed by probe. Token needs to map these two strings to the “1”b or “0”b values actually returned.

```
ON: call_priority

OFF: wakeup_control_points
```

These six specifications are built-in to the token command/af when the -dsav control argument is given.

Objective D was tested against many input streams to ensure the expected tokens were actually produced. This testing is simplified by the -mode input,specs control argument which causes token to display the tokens generated from the input string, and from the -match or -dsav specification control arguments.

Objectives E and **F** were tested by manual display of matching results for 51 elements displayed by the token command while implementing the event_channel_table.ec; and then returned by the token active function when running this exec_com.

ssu_request_tables_.alm

Changes to the ssu_request_tables_ added the token$token_request as a request in the exec_com_toolkit table. azm adds this table to its request table list so commands useful in exec_com scripts (like event_channel_table.azmec) are available for use in exec_coms.

```
azm:  lr -table exec_com_toolkit

Requests in table:  ssu_request_tables_$exec_com_toolkit
...
token   Extracts data items from an input string.
```

azm_requests_1_.pl1

The changes to add -in and -short (-sh) control arguments to the azm display request were tested manually by selecting a dump, and displaying the pds of any user process. The normal structure library used by the display request does not contain a definition for elements of the pds (process data segment) structure.

```
azm:  .
azm 2.5 (abbrev)
> ERF 011925.2154.0.1 in directory >old_dumps dumped at 01/19/25  2254.7 mst Sun.
Proc  20 DBR  16637154 running        on cpu a   Clayton.SysAdmin.a

azm:  d pds -as pds
azm (display): Cannot get library definition for pds
```

By using the -in and -long control args, information in the pds segment can be displayed as elements.

```
azm:   d pds -as pds -in >udd>m>gd>lib>st>pds -long

pds                           @ 71|0
  page_fault_data             @ 71|0
    prs (0) = 41|21766 bound_library_wired_$pl1_operators_|1426
    prs (1) = 230|36020 >sl1>stack_0.014|36020
    prs (2) = 46|14217 bound_wired_1$ocdcm_|13343
    prs (3) = 230|33740 >sl1>stack_0.014|33740
    prs (4) = 17|3210 ws_linkage|3210
    prs (5) = 70|1420 oc_data|1420
    prs (6) = 230|35700 >sl1>stack_0.014|35700
    prs (7) = 230|0 >sl1>stack_0.014
    regs                      @ 71|20
      x (0) = "014712"b3
      x (1) = "035640"b3
      x (2) = "000000"b3
      x (3) = "000460"b3
      x (4) = "000314"b3
      x (5) = "000000"b3
      x (6) = "000110"b3
      x (7) = "000120"b3
      a = "000002000010"b3
      q = "000000000112"b3
```

```
      e = "00"b4
      t = "000063016"b3
      ralr = "0"b3
...


azm:   d pds -as pds.page_fault_data.regs -in >udd>m>gd>lib>st>pds -short
    regs                @ 71|20
      x (0) = "014712"b3, x (1) = "035640"b3, x (2) = "000000"b3,
      x (3) = "000460"b3, x (4) = "000314"b3, x (5) = "000000"b3,
      x (6) = "000110"b3, x (7) = "000120"b3, a = "000002000010"b3,
      q = "000000000112"b3, e = "00"b4, t = "000063016"b3,
      ralr = "0"b3
...



azm:   d pds -as pds.page_fault_data.regs -in >udd>m>gd>lib>st>pds -sh
    regs                @ 71|20
      x (0) = "014712"b3, x (1) = "035640"b3, x (2) = "000000"b3,
      x (3) = "000460"b3, x (4) = "000314"b3, x (5) = "000000"b3,
      x (6) = "000110"b3, x (7) = "000120"b3, a = "000002000010"b3,
      q = "000000000112"b3, e = "00"b4, t = "000063016"b3,
      ralr = "0"b3
...
```

---

ipc_real_.pl1 and ipc_create_arg.incl.pl1

Changes to the ipc_$create_event_channel entry point of ipc_real_.pl1, and ipc_create_arg.incl.pl1 were tested using a small test program (ipcf.pl1).

This program:

- displays the location and value of ipc_data_$fast_channel_events, the external bit array containing the bit assigned to each fast event-wait channel assigned to the ECT in the current ring of execution of the user process;
- creates a fast event-wait channel, and uses new information returned by ipc_$create_event_channel to locate the particular fast_event_bit in the ipc_data_$fast_channel_event bit array associated with the new channel;
- verifies this fast_event_bit does get set when a wake-up is reported, and does get cleared by ipc_$block and ipc_$read_ev_chn when ipc_$block returns or ipc_$read_ev_chn returns.

It also verifies that it can check the fast_event_bit looking for a new wake-up, and can clear that bit before processing the wake-up all without calling either ipc_$block or ipc_$read_ev_chn.

ring0_get_.pl1

Changes to ring0_get_ code parameter declarations were checked using the display_entry_point_dcl (depd) command to display calling sequences of each of its entry points, as generated from the argument descriptors which precede each entry point.

Note that the last argument is now declared: fixed bin(35).

```
depd ring0_get_$(definition name names segptr)
dcl  ring0_get_$definition entry (ptr, char(*), char(*),
      fixed bin(18), fixed bin, fixed bin(35));
dcl  ring0_get_$name entry (char(*), char(*), ptr, fixed bin(35));
dcl  ring0_get_$names entry (char(*), ptr, ptr, fixed bin(35));
dcl  ring0_get_$segptr entry (char(*), char(*), ptr, fixed bin(35));
r 12:05 0.094 4
```

For the ring0_get_ entry points which accept `slt` pointers following the code argument, that code is also: fixed bin(35).

```
depd ring0_get_$(definition name segptr)_given_slt
dcl  ring0_get_$definition_given_slt entry (ptr, char(*), char(*),
      fixed bin(18), fixed bin, fixed bin(35), ptr, ptr, ptr);
dcl  ring0_get_$name_given_slt entry (char(*), char(*), ptr,
      fixed bin(35), ptr, ptr);
dcl  ring0_get_$segptr_given_slt entry (char(*), char(*), ptr,
      fixed bin(35), ptr, ptr);
r 12:05 0.065 3
```

pl1.dcl

Changes to pl1.dcl added declarations for missing ipc_ entry points: ipc_$read_ev_chn and ipc_$run_event_calls. These can now be displayed using the display_entry_point_dcl (depd) command.

```
depd ipc_$(read_ev_chn run_event_calls)
dcl  ipc_$read_ev_chn entry (fixed bin(71), fixed bin, ptr,
      fixed bin(35));
dcl  ipc_$run_event_calls entry (fixed bin, fixed bin(35));
```

Documentation

token.info

The new token command and active function is documented by the token.info segment shown below.

```
>udd>m>gd>w>ect>token.info   (184 lines in info)

2026-08-10  token

Syntax as a command:
   token -control_args

Syntax as an active function:
  [token -control_args]

Function:
This program splits an input string into tokens separated by whitespace
or comma characters (,).  It then compares these tokens to one or more
specifications.  If the initial set of tokens match a specification,
those tokens are selected for display.  If no matching tokens are
found, the initial token is removed and the comparisons are repeated.
This process continues until: a matching set of tokens is found; or the
input token list is exhausted.

The command displays the matching set of tokens, or issues a warning
that no matching set of tokens was found.

The active function returns the tokens which match a <value>
specification; or if no matching set was found, it returns "false".

Control arguments (input):
   Use one of the following control arguments to obtain a
   TOKEN_STRING which is parsed into tokens separated by whitespace.
-input TOKEN_STRING, -in TOKEN_STRING
   gives the string to be parsed.
-command COMMAND_LINE, -cmd COMMAND_LINE
   gives a single Multics command to be executed.  Its output is
   treated as the TOKEN_STRING to be parsed.  A typical command has the
   format:  -cmd "dump_segment ADDR -as STRUCTURE.ELEMENT"

Control arguments (specifications):
   One or more -match controls may be given to specify rules for token
   matching; or -dsav may be given; or one or more -match
   specifications may be given with -dsav.  Order of these controls
   determines sequence in which each rule is compared against the input
   tokens.  For details, see the "List of matching specifiers" section.
-match SPECIFICATION, -m SPECIFICATION
   gives a string defining one matching specification.  If -match is
   given several times, the first specification which matches the input
   tokens is used to select matching tokens.
```

```
-dump_segment_as_value, -dsav
   gives a group of predefined matching rules designed to extract a
   value from a structure element displayed by the Multics command:
       dump_segment ADDR -as STRUCTURE.ELEMENT
   These SPECIFICATIONS work best if the TOKEN_STRING is limited to
   tokens for only one structure element or one array element within
   the structure.  For more details, see "Notes on -display_as_value".

Control arguments (debugging):
-mode MODE_STRING, -md MODE_STRING
   sets the operational mode for token, according to the given
   MODE_STRING which is a string of mode names separated by commas.  No
   space characters appear in the MODE_STRING.  For more details, see
   the "List of mode names" section.

List of matching specifiers:
   A -match SPECIFICATION operand is a single string containing either
    - one or more character literals which must match an output token
      exactly; or
    - one or more <...> specifiers which can match any of a set of
      input token strings.  These specifiers are listed below.
<element_name>
   matches PL/I rules for a <reference> to a scalar element of a
   structure.  For example, the stack_header.ect_ptr <reference>
   identifies an ect_ptr scalar element of the stack_header structure.
   A TOKEN_STRING like "display 234 -as stack_header.ect_ptr" includes
   only the ect_ptr element name when displaying the value.  So
   <element_name> expects only a final element name in the token
   strings it tries to match.

<array_index>
   matches PL/I rules for a parenthesized-list of <subscript> strings.
   For example, a 1-dimensional array index has one subscript: (3).
   A 2-dimensional array index has two subscripts separated by a comma:
   (3, 5).  For matching purposes, any parenthesized-list token will
   match the <array_index> specifier.
<entrypoint_name>
   matches PL/I rules for a subroutine or function entrypoint name,
   which: may include both a ref_name and entrypoint separated by a
   dollar-sign ($) character; or may be just an element_name.

<skip>
   matches any token but does not return that matching token as the
   value of the token active string.
<skip: N>
   matches N consecutive tokens (where N is a positive integer) but
   does not return those matching tokens as part of the value of the
   token active string.

<value>
   matches any token which adheres to the rules for a PL/I character-
   or bit-string literal; or numeric literal; or pointer literal value.
   The token which matches this <value> specifier is returned as the
   value of the token active string.
<value: if <element_name>>
   matches any token accepted by <element_name>, but then acts like
   <value> to return that token.
```

```
<value: if <element_name> returns "1"b>
   matches any token accepted by <element_name>, but then acts like
   <value> that returns the bit-string: "1"b
<value: if <element_name> returns "0"b>
   matches any token accepted by <element_name>, but then acts like
   <value> that returns the bit-string: "0"b
<value: if <entrypoint_name>>
   matches any token accepted by <entrypoint_name>, but then acts like
   <value> to return that token.

List of mode names:
args
   display all input arguments.
input, in
   display output captured from -command or operand of -input
   after that string has been parsed into tokens.
specifications, specs, sp
   display match specifications after they have been parsed into
   tokens.
match, mh, m
   if the input tokens match one of the specifications and if token was
   not invoked via an active string, display tokens in the matched
   specification, and below each token display the matching input
   token.

Notes on -dump_segment_as_value:  Use of the -dsav control is
equivalent to the following six specifications:
 -m  "<element_name> <array_index> = <value> <value: if <entrypoint_name>>"
 -m  "<element_name> <array_index> = <value>"
 -m  "<element_name>               = <value> <value: if <entrypoint_name>>"
 -m  "<element_name>               = <value>"
 -m  "ON  : <value: if <element_name> returns "1"b>"
 -m  "OFF : <value: if <element_name> returns "0"b>"

Examples:
Value of a scalar structure element may be displayed as follows...

dump_segment 340|16040 -as ect_header.ect_areap
  ect_areap = 340|15770  >pdd>!zzzzzzzbBBBBBB>!BBBKWznZHhplJH.area.linker|15770

The following dump_segment command displays the tokens selected by one
of the -match specifications supplied using the -dsav control argument:

token -cmd "ds 340|16040 -as ect_header.ect_areap" -dsav
  ect_areap = 340|15770

The following active request returns only the value of this scalar
element:

string [token -cmd "ds 340|16304 -as ect_header.ect_areap" -dsav]
  340|15770

       ----------------------------------------
```

```
A more complex example involves an array element of a structure.  For
example:

dump_segment 340|16040 -as ect_header.count
  count (-1) = 0, count (0) = 95, count (1) = 41, count (2) = 53,
  count (3) = 1, count (4) and count (5) = 0

Because of the complex manner in which azm display can pool together
array elements with like values (such as count(4) and count(5)), it is
usually best to select only one array element by its index.  Notice
the use of brace characters surrounding the index number {4} so doubled
quoting to protect meaning of parentheses is not needed.

token -cmd "ds 340|16040 -as ect_header.count{4}" -dsav
  count (4) = 0

       ----------------------------------------

Displaying the value of an entrypoint pointer usually includes the name
of the called entrypoint following the procedure_ptr.  For example:

ds 247|20406 -as call_channel.procedure_ptr
  procedure_ptr = 323|7233   message_facility_$wakeup_processor

The following active function returns both the pointer value and
entrypoint name to replace the active string.  Because token rules are
designed to work within exec_com scripts (where &+ continuation lines
are easily added, the example below first saves the -match rule below
as an abbreviation.

.af MATCH_1 -match "<element_name> = <value> <value>"
r 18:28 0.015 2

.l MATCH_1
  MATCH_1  -match "<element_name> = <value> <value>"
r 18:29 0.061 1

string [token -cmd "ds 234 -as stack_header.ect_ptr" MATCH_1]
247|17240 >process_dir_dir>!BPLCjgfBBBBBBB
r 18:29 0.526 3
```

---

ssu.token.info

The token$token_request ssu_ request and active request entry point has slightly different documentation. The differences from the above command documentation are shown below.

```
cpa token.info ssu.token.info

A3        Syntax as a command:
Changed by B to:
B3        Syntax:


A7        Syntax as an active function:
Changed by B to:
B7        Syntax as an active request:
```

```
A20
A21       The command displays the matching set of tokens, or issues a warning
Changed by B to:
B20       The request displays the matching set of tokens, or issues a warning


A24       The active function returns the tokens which match a <value>
Changed by B to:
B23       The active request returns the tokens which match a <value>


A33       -command COMMAND_LINE, -cmd COMMAND_LINE
A34          gives a single Multics command to be executed.  Its output is
A35          treated as the TOKEN_STRING to be parsed.  A typical command has the
A36          format:  -cmd "dump_segment ADDR -as STRUCTURE.ELEMENT"
Changed by B to:
B32       -request REQUEST_LINE, -rq REQUEST_LINE
B33          gives a single azm request which will be executed.  Its output is
B34          treated as the TOKEN_STRING to be parsed.  A typical request has the
B35          format:  -rq "display ADDR -as STRUCTURE.ELEMENT"


A41          matching; or -dsav may be given; or one or more -match
A42          specifications may be given with -dsav.  Order of these controls
A43          determines sequence in which each rule is compared against the input
A44          tokens.  For details, see the "List of matching specifiers" section.
Changed by B to:
B40          matching; or -dav may be given; or one or more -match specifications
B41          may be given with -dav.  Order of these controls determines sequence
B42          in which each rule is compared against the input tokens.  For
B43          details, see the "List of matching specifiers" section below.


A51       -dump_segment_as_value, -dsav
A52          gives a group of predefined matching rules designed to extract a
A53          value from a structure element displayed by the Multics command:
A54              dump_segment ADDR -as STRUCTURE.ELEMENT
A55          These SPECIFICATIONS work best if the TOKEN_STRING is limited to
Changed by B to:
B50       -display_as_value, -dav
B51          gives a group of predefined matching rules designed to extract a
B52          value from a structure element displayed by the analyze_multics
B53          request:   display ADDR -as STRUCTURE.ELEMENT
B54
B55          These SPECIFICATIONS work best if the REQUEST_LINE is limited to


A78          A TOKEN_STRING like "display 234 -as stack_header.ect_ptr" includes
Changed by B to:
B78          A REQUEST_LINE like "display 234 -as stack_header.ect_ptr" includes


A128         display output captured from -command or operand of -input
Changed by B to:
B128         display output captured from -request_line or operand of -input


A140      Notes on -dump_segment_as_value:  Use of the -dsav control is
A141      equivalent to the following six specifications:
Changed by B to:
B140      Notes on -display_as_value:  Use of the -dav control is equivalent to
B141      the following six specifications:
```

```
A153      dump_segment 340|16040 -as ect_header.ect_areap
Changed by B to:
B153      azm:  d 340|16040 -as ect_header.ect_areap


A156      The following dump_segment command displays the tokens selected by one
A157      of the -match specifications supplied using the -dsav control argument:
A158
A159      token -cmd "ds 340|16040 -as ect_header.ect_areap" -dsav
Changed by B to:
B156      The following azm request displays the tokens selected by one of the
B157      -match specifications supplied using the -dav control argument:
B158
B159      azm: token -rq "d 340|16040 -as ect_header.ect_areap" -dav


A166      string [token -cmd "ds 340|16304 -as ect_header.ect_areap" -dsav]
Changed by B to:
B166      azm:  sbag &1 [token -rq "d 340|16304 -as ect_header.ect_areap" -dav]


A176      dump_segment 340|16040 -as ect_header.count
Changed by B to:
B176      azm:  d 340|16040 -as ect_header.count


A182      usually best to select only one array element by its index.  Notice
A183      the use of brace characters surrounding the index number {4} so doubled
Changed by B to:
B182      usually best to select only one array element by its index.  Notice the
B183      use of brace characters surrounding the index number {4} so doubled


A186      token -cmd "ds 340|16040 -as ect_header.count{4}" -dsav
Changed by B to:
B186      azm:  token -rq "d 340|16040 -as ect_header.count{4}" -dav


A193      Displaying the value of an entrypoint pointer usually includes the name
A194      of the called entrypoint following the procedure_ptr.  For example:
A195
A196      ds 234 -as stack_header.ect_ptr
Changed by B to:
B193      Displaying the value of an entry variable usually includes the name of
B194      the called entrypoint following the procedure_ptr.  For example:
B195
B196      azm:  d 234 -as stack_header.ect_ptr


A199
A200      The following token active function returns both the pointer value and
Changed by B to:
B199      The following token active request returns both the pointer value and


A203      are easily added, the example below first saves the -match rule as an
A204      abbreviation.
Changed by B to:
B202      are easily added, the example below first saves the -match rule below
B203      as an abbreviation.  The abbreviation system must be turned on in azm
B204      requests.

Comparison finished: 19 differences, 80 lines.
```

ipc_.info

Several changes were made in ipc_.info relating to the ipc_$block, ipc_$create_event_channel, ipc_$read_ev_chn, ipc_$run_event_calls, ipc_$set_call_prior and ipc_$set_wait_prior entry points. These are shown in the compare_ascii output below.

```
compare_ascii >doc>info>ipc_.info ==

A1        :Info: ipc_:  2025-02-07  ipc_
Changed by B to:
B1        :Info: ipc_:  2026-08-27  ipc_


A44       :Entry:  block:  1982-10-22 ipc_$block
Changed by B to:
B44       :Entry:  block:  2026-08-27 ipc_$block


A59          channels on which events are being awaited.  (Input) This structure
A60          is declared in event_wait_list.incl.pl1. Frequently ipc_$block is
A61          called with only one channel in the wait list.  In this case, the
A62          event_wait_channel structure can be used (declared in
A63          event_wait_channel.incl.pl1).
A64       event_wait_info_ptr
A65          is a pointer to the event_wait_info structure into which the
A66          ipc_$block entry point can put information about the event that
A67          caused it to return (i.e., that awakened the process).  This
A68          structure is declared in event_wait_info.incl.pl1.  (Input)
A69       code
A70          is a standard status code.  (Output)
A71
A72
A73       :Entry:  create_event_channel:  2025-02-07  ipc_$create_event_channel
Changed by B to:
B59          channels on which events are being awaited. (Input)  This structure
B60          is declared in event_wait_list.incl.pl1. Frequently ipc_$block is
B61          called with only one channel in the wait list.  In this case, use
B62          the event_wait_channel structure declared in
B63          event_wait_channel.incl.pl1.
B64       event_wait_info_ptr
B65          is a pointer to the event_wait_info structure into which the
B66          ipc_$block entry point can put information about the event that
B67          caused it to return (i.e., that awakened the process).  This
B68          structure is declared in event_wait_info.incl.pl1.  (Input)
B69       code
B70          is a standard status code.  (Output)
B71
B72
B73       Notes on the event_wait_list structure:
B74       The include segment event_wait_list.incl.pl1 declares the following
B75       structure.  If you wish to monitor more than one event-wait channel,
B76       pass the address of this structure as the event_wait_list_ptr argument
B77       in the call to ipc_$block.
B78
B79
B80          dcl event_wait_list_n_channels
B81                                fixed binary;
B82          dcl event_wait_list_ptr pointer;
B83
```

```
B84           dcl 1 event_wait_list   aligned based (event_wait_list_ptr),
B85               2 n_channels        fixed binary,
B86               2 pad               bit (36),
B87               2 channel_id        (event_wait_list_n_channels
B88                                    refer (event_wait_list.n_channels))
B89                                   fixed binary (71);
B90
B91
B92        List of event_wait_list elements:
B93        event_wait_list_n_channels
B94           set this variable to the number of event-wait and fast event-wait
B95           channels you wish to monitor.
B96        event_wait_list_ptr
B97           points to locate at which the event_wait_list was allocated.
B98        event_wait_list
B99           allocate the adjustable array in the area of your choice.  The
B100          allocate operation sets the event_wait_list_ptr and the n_channels
B101          element.
B102       channel_id
B103          set the channel_id elements in the array to the channel_name of the
B104          event-wait or fast event-wait to be monitored.
B105
B106
B107       Notes on the event_wait_channel structure:
B108       The include segment event_wait_channel.incl.pl1 declares the following
B109       structure.  If you wish to monitor only one event-wait channel, pass
B110       the address of this structure as the event_wait_list_ptr argument in
B111       the call to ipc_$block.
B112
B113          dcl 1 event_wait_channel   aligned automatic,
B114              2 n_channels           fixed bin initial (1),
B115              2 pad                  bit (36),
B116              2 channel_id           (1) fixed bin (71);
B117
B118
B119       List of event_wait_channel elements:
B120       channel_id
B121          set the channel_id element to the channel_name of the event-wait or
B122          fast event-wait channel to be monitored.
B123
B124
B125       Notes on the event_wait_info structure:
B126       The include segment event_wait_info.incl.pl1 declares the following
B127       structure.  Give the address of this structure as the
B128       event_wait_info_ptr argument to ipc_$block.  Elements of this structure
B129       will be returned by ipc_$block to identify the event_channel which
B130       received an event.
B131
B132          dcl 1 event_wait_info aligned automatic,
B133              2 channel_id      fixed bin(71),
B134              2 message         fixed bin(71),
B135              2 sender          bit(36),
B136              2 origin,
B137                3 dev_signal    bit(18) unaligned,
B138                3 ring          fixed bin(17) unaligned,
B139              2 channel_index   fixed bin;
B140
B141
B142       List of event_wait_info elements:
B143       channel_id
B144          name of the event-wait or fast event-wait channel that received an
B145          event.
```

```
B146      message
B147         if channel_id names an event-wait channel, this is the 2-word
B148         message passed in the call to hcs_$wakeup notifying that an event
B149         occurred on the given channel.  If channel_id names a fast
B150         event-wait channel, this element is not set.  Fast event-wait
B151         channels receive no message with their wake-up.
B152      sender
B153         process_id of the process which called hcs_$wakeup.
B154
B155
B156      dev_signal
B157         is "1"b if the wakeup was generated by a hardware device interrupt.
B158         if "0"b if the wakeup was requested by a cooperating user or system
B159         process.
B160      ring
B161         is the ring of execution of the process which sent the wake-up.
B162      channel_index
B163         index at which channel_id occurred in the event_wait_list or
B164         event_channel_info structure.
B165
B166
B167      :Entry:  create_event_channel:  2026-08-27  ipc_$create_event_channel


A89          pointer to an ipc_create_arg_structure describing requirements for
A90          the new IPC channel. (Input)  For details, see the "Notes on the
A91          ipc_create_arg_structure" section below.
Changed by B to:
B183         points to one of the structures describing requirements for
B184         the new IPC channel. (Input)  For details, see the "Notes on
B185         creating an event channel" section below.


A98       Notes on the ipc_create_arg_structure:
A99       Input data for this subroutine is provided using the ipc_create_arg
A100      structure described below.  A declaration for this structure and
A101      related constants is provided in the ipc_create_arg.incl.pl1 segment.
A102      Each element of the structure is described in the "List of
A103      ipc_create_arg elements" section below.
A104
A105      dcl  1 ipc_create_arg_structure
A106                              aligned based (ipc_create_arg_structure_ptr),
A107           2 version          char (8) unaligned,
A108           2 channel_type     fixed bin,
A109           2 call_entry       variable entry (ptr),
A110           2 call_data_ptr    ptr,
A111           2 call_priority    fixed bin (17);
A112
A113
A114      dcl  ipc_create_arg_structure_ptr    ptr;
A115
A116      dcl (ipc_create_arg_structure_v1     char (8)init ("ipcarg01"),
Changed by B to:
B192      Notes on creating an event channel:
B193      Input data for this subroutine is provided using either the
B194      ipc_create_wait_event, ipc_create_fast_event or the
B195      ipc_create_arg_structure described below.  A declaration for these
B196      structures and related constants is provided in the include segment
B197      ipc_create_arg.incl.pl1.
B198
B199      dcl (ipc_create_arg_structure_v1     char (8)   init ("ipcarg01"),
```

```
Inserted in B:
B207      dcl  1 ipc_create_wait_event
B208                              aligned automatic,
B209           2 version          char (8) unaligned
B210                                        init(ipc_create_arg_structure_v1),
B211           2 channel_type     fixed bin  init(WAIT_EVENT_CHANNEL_TYPE),
B212           2 pad1             fixed bin,
B213           2 pad2             variable entry (ptr),
B214           2 pad3             ptr,
B215           2 pad4             fixed bin,
B216           2 pad5             fixed bin;
B217
B218
B219      dcl  1 ipc_create_fast_event
B220                              aligned automatic,
B221           2 version          char (8) unaligned
B222                                        init(ipc_create_arg_structure_v1),
B223           2 channel_type     fixed bin  init(FAST_EVENT_CHANNEL_TYPE),
B224           2 pad1             fixed bin,
B225           2 pad2             variable entry (ptr),
B226           2 event_bit_ptr    ptr        init(null()),
B227           2 pad4             fixed bin,
B228           2 pad5             fixed bin,
B229
B230           fast_event_bit     bit(1) unaligned
B231                              based (ipc_create_fast_event.event_bit_ptr);
B232
B233
B234      dcl  1 ipc_create_arg_structure
B235                              aligned based (ipc_create_arg_structure_ptr),
B236           2 version          char (8) unaligned,
B237           2 channel_type     fixed bin,
B238           2 pad1             fixed bin,
B239           2 call_entry       variable entry (ptr),
B240           2 call_data_ptr    ptr,
B241           2 call_priority    fixed bin (17),
B242           2 pad5             fixed bin,
B243
B244           ipc_create_arg_structure_ptr
B245                              ptr;
B246
B247
B248      List of ipc_create_wait_event elements:
B249         The following items are elements of the ipc_create_wait_event
B250         structure shown in the preceding section of this info segment.
B251         The pad elements are ignored when creating a wait channel.
B252      version
B253         gives the version number of this structure. (Input)  It is
B254         initialized to the value in ipc_create_arg_structure_v1.
B255      channel_type
B256         gives the type of IPC event channel to be created. (Input)  It is
B257         initialized to:  WAIT_EVENT_CHANNEL_TYPE.
B258
B259
B260      List of ipc_create_fast_event elements:
B261         The following items are elements of the ipc_create_fast_event
B262         structure shown in the "Notes on creating an event channel" section
B263         above.  The pad elements are ignored when creating a fast channel.
B264      version
B265         gives the version number of this structure. (Input)  It is
B266         initialized to the value in ipc_create_arg_structure_v1.
B267      channel_type
B268         gives the type of IPC event channel to be created. (Input)  It is
B269         initialized to:  WAIT_EVENT_CHANNEL_TYPE.
```

```
B270      event_bit_ptr
B271         points to the fast_event_bit assigned to the created fast channel.
B272         (Output)
B273
B274
B275      fast_event_bit
B276         This bit may be tested to see if the fast event received a
B277         wake-up while the process was blocked.
B278          "1"b - one or more events were received.
B279          "0"b - no event has been received yet.
B280
B281
Preceding:
A124      List of ipc_create_arg_structure elements:


A126         shown in the preceding section of this info segment.  The
A127         call_entry, call_data_ptr and call_priority must be given for an
A128         event call channel; but are ignored when creating an event wait
A129         channel.
A130      version
A131         gives the version number of this structure. (Input)  Set it to
A132         the constant value ipc_create_arg_structure_v1 in the include
A133         segment.
A134      channel_type
A135         gives the type of IPC event channel to be created. (Input)  See
A136         the "List of channel types" section below.
Changed by B to:
B284         shown in the "Notes on creating an event channel" section above.
B285      version
B286         gives the version number of this structure. (Input)  It is
B287         initialized to the value in ipc_create_arg_structure_v1.
B288      channel_type
B289         gives the type of IPC event channel to be created. (Input)  Set it
B290         to one of the constants:  CALL_EVENT_CHANNEL_TYPE, or
B291         ASYNC_CALL_EVENT_CHANNEL_TYPE.


A150         with the lowest priority number is called before other pending
A151         channels with higher numbers.
Changed by B to:
B305         with the lowest priority number is processed before other pending
B306         channels with larger numbers.


A158      FAST_EVENT_CHANNEL_TYPE
A159         create a fast event wait channel.
Deleted by B, preceding:
B313      WAIT_EVENT_CHANNEL_TYPE


Inserted in B:
B315      FAST_EVENT_CHANNEL_TYPE
B316         create a fast event wait channel.
Preceding:
A162
A163      CALL_EVENT_CHANNEL_TYPE


A339      :Entry:  read_ev_chn:  1982-10-22 ipc_$read_ev_chn
Changed by B to:
B494      :Entry:  read_ev_chn:  2026-08-27 ipc_$read_ev_chn
```

```
A355         is the identifier of the event channel.  (Input)
Changed by B to:
B510         is the identifier of the event channel to be read.  (Input)


Inserted in B:
B525      Notes on the event_wait_info structure:
B526      The include segment event_wait_info.incl.pl1 declares the following
B527      structure.  Give the address of this structure as the
B528      event_wait_info_ptr argument to ipc_$block.  Elements of this structure
B529      will be returned by ipc_$block to identify the event_channel which
B530      received an event.
B531
B532         dcl 1 event_wait_info aligned automatic,
B533             2 channel_id      fixed bin(71),
B534             2 message         fixed bin(71),
B535             2 sender          bit(36),
B536             2 origin,
B537               3 dev_signal    bit(18) unaligned,
B538               3 ring          fixed bin(17) unaligned,
B539             2 channel_index   fixed bin;
B540
B541
B542      List of event_wait_info elements:
B543      channel_id
B544         name of the event-wait or fast event-wait channel that received an
B545         event.
B546      message
B547         if channel_id names an event-wait channel, this is the 2-word
B548         message passed in the call to hcs_$wakeup notifying that an event
B549         occurred on the given channel.  If channel_id names a fast
B550         event-wait channel, this element is not set.  Fast event-wait
B551         channels receive no message with their wake-up.
B552      sender
B553         process_id of the process which called hcs_$wakeup.
B554
B555
B556      dev_signal
B557         is "1"b if the wakeup was generated by a hardware device interrupt.
B558         if "0"b if the wakeup was requested by a cooperating user or system
B559         process.
B560      ring
B561         is the ring of execution of the process which sent the wake-up.
B562      channel_index
B563         is not set by ipc_$read_ev_chn.
B564
B565
Preceding:
A370      :Entry:  reconnect:  1982-10-22 ipc_$reconnect


A392      :Entry:  run_event_calls:  2025-02-07  ipc_$run_event_calls
Changed by B to:
B588      :Entry:  run_event_calls:  2026-08-27  ipc_$run_event_calls


A399      pending event calls in the process.
A400
A401
A402      Syntax:
A403      declare ipc_$run_event_calls (fixed bin(71), fixed bin(35));
A404      call ipc_$run_event_calls (channel_id, code);
A405
A406
```

```
A407      Arguments:
A408      channel_id
A409         is the identifier of the event call channel whose pending event call
A410         wakeups are to be processed. (Input)
Changed by B to:
B595      pending asynchronous event call channels in the process.
B596
B597
B598      Syntax:
B599      declare ipc_$run_event_calls (fixed bin, fixed bin(35));
B600      call ipc_$run_event_calls (channel_type, code);
B601
B602
B603      Arguments:
B604      channel_type
B605         is the type of event channel to run. (Input)  For details, see the
B606         "List of channel types" section.


A415      :Entry:  set_call_prior:  1982-10-22 ipc_$set_call_prior
Changed by B to:
B611      List of channel types:
B612         For details on the various types of event channels, type:
B613            help event_channel.gi
B614         The ipc_create_arg.incl.pl1 include segment declares the following
B615         constants.
B616
B617      CALL_EVENT_CHANNEL_TYPE (=3)
B618         run only regular event call channels.
B619      ASYNC_CALL_EVENT_CHANNEL_TYPE (=4)
B620         run only asynchronous event call channels.
B621      ANY_CALL_EVENT_CHANNEL_TYPE (=10)
B622         run both types of event call channels.
B623
B624
B625      :Entry:  set_call_prior:  2026-08-27 ipc_$set_call_prior


A422      affected.  By default, event-call channels have priority.
Changed by B to:
B632      affected.  By default, event wait channels have priority.


A435      :Entry:  set_wait_prior:  1982-10-22 ipc_$set_wait_prior
Changed by B to:
B645      :Entry:  set_wait_prior:  2026-08-27 ipc_$set_wait_prior


A442      current ring are affected.
Changed by B to:
B652      current ring are affected.  By default, event wait channels have
B653      priority.


A481                                                        END HISTORY COMMENTS */
A482
Changed by B to:
B692        2) change(2026-08-27,GDixon):
B693           A) Indicate that event wait_channels have priority by default in
B694              descriptions of ipc_$set_wait_prior and ipc_$set_call_prior.
B695           B) Correct argument descriptions in ipc_$run_event_calls.
B696           C) Add description of structure elements passed to ipc_$block
B697              and ipc_$read_ev_chn.
B698                                                        END HISTORY COMMENTS */

Comparison finished: 20 differences, 366 lines.
```

event_channels.gi.info

Changes made to this description of event channels are shown in the compare_ascii output below.

```
----------  event_channels.gi.info

compare_ascii >doc>info>event_channels.gi.info ==

A1        07/11/86  Event channels
A2
A3        Event channels can be thought of as numbered slots in the interprocess
A4        communication facility tables.  Each channel is either an event-wait,
A5        event-call, or asynchronous event-call channel.  An event-wait channel
A6        receives events that are merely marked as having occurred and awakens
A7        the process if it is blocked waiting for an event on that channel.  On
A8        an event-call channel, the occurrence of an event causes a specified
A9        procedure to be called if (or when) the process is blocked waiting for
A10       an event on any channel.  On an asynchronous event-call channel, the
A11       occurrence of an event causes a specified procedure to be called
A12       whether or not the process is blocked.  Naturally, the specific event
A13       channel must be made known to the process that expected to notice the
A14       event.  For an event to be noticed by an explicitly cooperating
A15       process, the event channel identifier value is typically placed in a
A16       known location of a shared segment.  For an event to be noticed by a
A17       system module, a subroutine call is typically made to the appropriate
A18       system module.  A process can go blocked waiting for an event to occur
A19       or can explicitly check to see if it has occurred.  If an event occurs
A20       before the target process goes blocked, then it is immediately awakened
A21       when it does go blocked.
A22
A23
A24       The user can operate on an event channel only if his ring of execution
A25       is the same as his ring when the event channel was created.
Changed by B to:
B1        :Info:  event_channels.gi:  event_channel.gi:  2026-08-27  Event channels
B2
B3        Event channels can be thought of as named slots in the Inter-Process
B4        Communication (IPC) facility tables.  Each channel is either an
B5        event-wait, fast event-wait, event-call, or asynchronous event-call
B6        channel.
B7
B8
B9        An event-wait channel receives events that are recorded as having
B10       occurred.  The process owning the channel is awakened if it called
B11       ipc_$block to wait for an event on that channel.  Or the owning process
B12       can call ipc_$read_ev_chn to check if the channel has received an
B13       event.  An event_wait_info structure is returned to the caller by both
B14       of these subroutines.
B15
B16       A fast event-wait channel operates like a regular event-wait channel,
B17       but has no event record associated with each event.  Instead there is
B18       one bit in which to record that one or more events have been reported.
B19       Both ipc_$block and ipc_$read_ev_chn will clear this bit when they
B20       report occurrence of the fast event-wait.  Or the caller can quickly
B21       monitor and clear this event bit directly.  It is one of 36 event bits
B22       stored in ipc_data_$fast_channel_events external static variable.
B23
B24
B25       For an event-call channel, the occurrence of an event causes a
B26       specified procedure to be called to handle each event when the process
B27       is blocked waiting for a event-wait on any channel.  By default, an
B28       ipc_$block looks for notified event-call channel only if none of the
B29       event-wait channels in its wait-list have received an event.
```

B30
B31 An asynchronous event-call channel operates like a regular event-call
B32 channel except that occurrence of its event causes its handling
B33 procedure to be called immediately: it does not wait for the owning
B34 process to become blocked. On-going work in that process is
B35 interrupted by a wkp_ Inter-Process Signal. The wkp_signal_handler_
B36 calls ipc_$run_event_calls to process any asynchronous event-call
B37 channels that have received an event.
B38
B39
B40 For each event type, information about the event channel must be made
B41 known to the process that is expected to notice the event. For an
B42 event to be noticed by an explicitly cooperating process, information
B43 about the channel is typically placed in a known location of a shared
B44 segment. This includes the event channel name, process_id of the
B45 owner, and expected content of a 2-word event message that accompanies
B46 most event notifications. For an event to be noticed by a system
B47 module, a subroutine call is typically made to convey this information
B48 to the appropriate system module.
B49
B50
B51 The owning process can call ipc_$read_ev_chn to check if a channel
B52 event has occurred; that subroutine returns immediately with
B53 information about the next event reported for that channel; or with a
B54 status code reporting no event found. Or the owning process can call
B55 ipc_$block waiting for an event. If an event-wait channel in the
B56 ipc_$block wait-list receives an event before the target process goes
B57 blocked, then ipc_$block returns immediately with information for the
B58 first notified event in the wait-list.
B59
B60
B61 The owning process can operate on an event channel only when his ring
B62 of execution is the same as his ring in which the event channel was
B63 created.

A28 interprocess communication facility. The hcs_$wakeup entry point is
A29 used to wake up a blocked process for a specified event.
A30
A31
A32 Invoking an Event-Call Procedure: When a process is awakened on an
A33 event-call channel, control is immediately passed to the procedure
A34 specified by the input arguments to the ipc_$create_event_channel entry
A35 point. The procedure is called with one argument, a pointer to the
A36 following structure. This structure is declared in
A37 event_call_info.incl.pl1.
Changed by B to:
B66 Inter-Process Communication (IPC) facility. The hcs_$wakeup entry
B67 point is used to notify the owning process that a specified event has
B68 occurred.
B69
B70
B71 Notes on an event-call procedure:
B72 When a process is awakened on an event-call channel, control is
B73 immediately passed to the handling procedure specified in the
B74 call to ipc_$create_event_channel. The procedure is called
B75 with one argument, a pointer to the following structure. This
B76 structure is declared in event_call_info.incl.pl1.
A50 Structure elements:
A51 channel_id
A52 is the identifier of the event channel.
A53 message
A54 is an event message as specified to the hcs_$wakeup entry point.
A55 sender
A56 is the process identifier of the sending process.

```
A57          dev_signal
A58             indicates whether the event occurred as the result of an I/O
A59             interrupt.
A60             "1"b yes
A61             "0"b no
A62          ring
A63             is the sender's validation level.
A64          data_ptr
A65             points to further data to be used by the called procedure.
A66
A67
A68       Notes: A user should be familiar with interprocess communication in
Changed by B to:
B89       List of event_call_info elements:
B90       channel_id
B91          is the identifier of the event channel.
B92       message
B93          is an event message as specified to the hcs_$wakeup entry point.
B94          The owner of an event-channel must define content of such message
B95          words when a specific event-channel data is make known to the
B96          process expected to notice the event.
B97       sender
B98          is the process identifier of the sending process.
B99
B100
B101      dev_signal
B102         indicates whether the event occurred as the result of an I/O
B103         interrupt.
B104           "1"b yes
B105           "0"b no
B106      ring
B107         is the sender's validation level.
B108      data_ptr
B109         is an optional pointer argument provided when the event-call channel
B110         was created.  If non-null, it points to owner-provided information
B111         needed by the handling procedure to process the event-call message.
B112
B113
B114      Notes:  A user should be familiar with Inter-Process Communication in


A76       If a program establishes an event-call channel, and the procedure
A77       associated with the event-call channel uses static storage, then the
A78       event-call procedure should have the perprocess_static attribute.
A79       This is not necessary if the procedure is part of a limited subsystem
A80       in which run units cannot be used.  See the description of the run
A81       command for more information on run units and perprocess_static.
Changed by B to:
B122
B123      If a program establishes an event-call channel, and the handling
B124      procedure associated with the event-call channel uses static storage,
B125      then the event-call procedure should have the option(separate_static)
B126      attribute in the main procedure statement of the source containing the
B127      handling procedure.  See the description of the option(separate_static)
B128      attribute in the Multics PL/I Language Specification (AG94) manual.
B129
B130      Use of an option(separate_static) is not necessary if the procedure is
B131      part of a limited subsystem in which run units cannot be used.  See the
B132      description of the run command in the MPM Commands (AG92) manual for
B133      more information on run units and perprocess_static
```

```
B134
B135
B136      :hcom:
B137      /****^  HISTORY COMMENTS:
B138        1) change(2026-08-27,GDixon):
B139           Update info segment with better descriptions of event channel types, and
B140           corrected Notes section on coding for asynchronous event-call handling
B141           procedures.
B142                                                        END HISTORY COMMENTS */

Comparison finished: 4 differences, 181 lines.
```

Version History

Date	Revision	Author	Comment
2026-08-27	0.1	Gary Dixon	Pre-release versions of document.
2026-08-27	0.2	Gary Dixon	Use event_channels.gi.info description of event channel types.
2026-09-12	1.0	Gary Dixon	Publication version of document.
2026-09-13	1.1	Gary Dixon	Correct changes to ipc_.info segment.