z/TPF MEM5 Driver

User's Guide

This page intentionally left blank.

ZTEST MEM5

The MEM5 driver tests ECB and system heap test cases. This driver also runs test cases for grouped memory functional interfaces.

Requirements and restrictions

None.

Format

```
                             .- -HELP-----------------------------------.
>>--ZTEST--+-----+-- --MEM5--+------------------------------------------+-><
           +- -i-+           |                                          |
           '- -*-'           +-+- -API-name-+--+----------------------+-+
                             | '- -API-ALL--'  '-| MEM5 Parameters  |-' |
                             |                                          |
                             '- -?--------------------------------------'

MEM5 Parameters:

  .- -NOERR-.  .- -Verbose---.  .- -CNT-1--------.
|-+---------+--+-------------+--+----------------+------------------------>
  '- -ERR---'  '- -NOVerbose-'  +- -CNT-count----|
                                '- -TIme-seconds-'
```

i

indicates the specific I-stream in which the driver will be run. If *i* is not specified, the test case(s) will be executed on the I-stream on which the command is entered.

*****

specifies the driver will be invoked on all currently defined and available I-streams.

API-*name*

specifies which application program interface (API) to run, where *name* must be one of the following supported APIs:

- EHEAP
- SHEAP
- SHEAP2
- SWB
- MIXED
- LONG
- MALL
- MAX

API-ALL

specifies that all APIs supported by this driver will be run.

ERR

exercises the error test cases.

NOERR
exercises the normal test cases. NOERR is the default.

Verbose
displays all the informational messages. Verbose is the default.

NOVerbose
displays only error messages.

CNT-*count*
specifies the number of times to run the request, where *count* can be 1 to 5 digits. The default value for *count* is 1. CNT and TIME are mutually exclusive.

TIme-*seconds*
specifies the amount of time in seconds to run the request, where *seconds* can be 1 to 4 digits. CNT and TIME are mutually exclusive.

HELP | ?
HELP, ?, or entering an incorrect format will cause a display describing the format of the ZTEST MEM5 command. A list of supported APIs is included in the help display.

Source code information

The MEM5 driver consists of the following program segments:

Header Files

Header File	Description
qm5a70.h	Contains prototypes and various #defines.
qm5a80.h	Contains a table of API parameter permutations.

Macros

None.

BSOs

None.

CSOs

Module	Makefile	Segment	Description
QM5A	qm5a.mak	qm5a00.c	This segment is the main entry point for the MEM5 driver. It will perform parsing and invoke the appropriate routine.
		qm5a70.c	This segment contains MEM5 driver tests: • ECB Heap • SWBs • Mixed memory usage: - ECB Heap - system heap - SWBs - C++ new and delete operators - traditional storage (getcc and relcc) • Long running, looping tests: - ECB Heap - system heap • ECB Max • Mallinfo
		qm5a80.c	This segment contains system heap tests.
QM5B	qm5b.mak	qm5b.c	This segment contains ECB Heap and system heap test that require a second ECB.

Additional information

The API names specify a group of test cases. The following section provides more information about how the APIs are functionally grouped.

Test Case Coverage for "EHeap" (Where Noted)

Test cases for the detection of the corruption in the ECB heap (CTL-75).

Test cases for user-definable buffer sizes for ECB heap available lists (AVL1-AVL4).

Normal and error test cases for ECB Heap Management APIs: tpf_eheap_tag( ) and tpf_eheap_locate( ).

Normal cases for malloc API:

- request size of 1
- request small size heap
- request medium size heap
- request large size heap
- request > pre-allocated size
- request maximum amount of ECB heap
- change maximum number of frames, request more heap
- read and write to ECB malloc heap

Normal cases for calloc API:

- request size of 1, number elements 1
- request size of 5, number elements 5

- request small size heap
- request medium size heap
- request large size heap
- request array of char pointers
- request > pre-allocated size
- request maximum amount of ECB heap
- change maximum number of frames, request more heap
- read and write to ECB calloc heap
- verify calloc area is initialized to zero

Normal cases for realloc API:

- malloc, realloc size of 1
- malloc, realloc larger size
- malloc, realloc with a smaller size
- calloc, realloc with exact same size
- malloc, realloc with NULL pointer
- calloc, write to storage, realloc larger size, read from storage
- calloc, realloc > pre-allocated size
- request maximum amount of ECB heap
- change maximum number of frames, realloc larger
- read and write to ECB realloc heap
- malloc 31 bit, realloc 64 bit
- malloc 64 bit, realloc 31 bit
- calloc 31 bit, realloc 64 bit
- calloc 64 bit, realloc 31 bit

Normal cases for free API:

- free storage from malloc
- free storage from calloc
- free storage from realloc
- free NULL pointer
- free address of 0

Normal cases for exit API: (EHeap2)

- exit with different return codes
- exit with -1
- exit with maximum value

Error cases for malloc API:

- request size 0
- request size -1
- request size of "maximum integer" (2,147,483,647)
- access bytes prior to malloc'd memory beginning address
- access bytes following malloc'd memory ending address

Error cases for calloc API:

- request size 0, number elements 0
- request size 1, number elements 0
- request size -1, number elements 1
- request size of "maximum integer" (2,147,483,647)

Error cases for realloc API:

- request size -1
- request size 0, pointer not NULL
- request size of "maximum integer" (2,147,483,647)
- malloc, free malloc pointer realloc with freed pointer
- request with invalid address

Error cases for free API:

- malloc, realloc and free original malloc pointer
- free invalid address of -1
- free embedded address of malloc'd area

Test Case Coverage for "SHeap" and "SHeap2" (Where Noted)

Normal cases for gsysc API:

- request 1 frame
- request a small number of frames
- request a medium number of frames
- request a large number of frames
- two requests for the same amount of frames with the same tokens
- read and write to system heap from the requesting ECB
- read and write to system heap from a second ECB ("SHeap2")
- token of 8 nulls
- token with embedded null
- token with initial null
- not null terminated 8 byte token (that is, a 9 byte token)

Normal cases for tpf_gsysc API:

- request 1 frame
- request a small number of frames *
- request a medium number of frames
- request a large number of frames
- two requests for the same amount of frames with the same tokens * (if unique token path then only one is requested as unique)
- read and write to system heap from the requesting ECB
- read and write to system heap from a second ECB ("SHeap2")
- token of 8 nulls *
- token with embedded null *
- token with initial null *
- not null terminated 8 byte token (that is, a 9 byte token) *

  * also a tpf_fsysc test case if a unique token requested on the tpf_gsysc call

Normal cases for tpf_fsysc API (these only apply to "unique token" tpf_gsysc's):

- any tests in the tpf_gsysc section above that are flagged by an asterisk.
- find token from a second ECB ("SHeap2")

Normal cases for rsysc API:

- return gsysc'd system heap by address
- return tpf_gsysc'd system heap by address

Normal cases for tpf_rsysc API:

- return tpf_gsysc'd system heap by address
- return tpf_gsysc'd system heap by token
- return tpf_gsysc'd system heap from the requesting ECB
- return tpf_gsysc'd system heap from a second ECB ("SHeap2")
- return gsysc'd system heap from a second ECB using tpf_rsysc ("SHeap2")

Error cases for gsysc API:

- request zero frames
- request -1 frames
- request "maximum integer" number of frames (2,147,483,647)
- request same unique token twice (applies only if unique token requested)

Error cases for tpf_fsysc API:

- find of nonexistent token
- token pointer = -1
- find of released token
- find on non-unique token

Error cases for rsysc API:

- address of zero
- address of -1
- wrong (but valid) address
- release the same address twice
- access system heap after returning it ("SHeap2")

Error cases for tpf_rsysc API:

- address of zero
- address of -1
- wrong (but valid) address
- token of 8 nulls (when original request had a non-null token)
- wrong (but valid) token
- release the same address twice
- release the same unique token twice
- access system heap after returning it

("SHeap2", not all combinations done since expected result is a CTL-4 dump, two second sleep done between entries to slow down dumps)

Unless otherwise noted the above tpf_gsysc, tpf_fsysc, and tpf_rsysc cases are done for each possible combination of parameters:

frames	**BIT**	**owner**	**token**
4K	31-bit	no	non-unique
4K	31-bit	no	unique
4K	31-bit	yes	non-unique
4K	31-bit	yes	unique
4K	64-bit	no	non-unique
4K	64-bit	no	unique
4K	64-bit	yes	non-unique
4K	64-bit	yes	unique

frames	BIT	owner	token
1MB	31-bit	no	non-unique
1MB	31-bit	no	unique
1MB	31-bit	yes	non-unique
1MB	31-bit	yes	unique
1MB	64-bit	no	non-unique
1MB	64-bit	no	unique
1MB	64-bit	yes	non-unique
1MB	64-bit	yes	unique

Note: The following owner string types are used with at least one permutation of tpf_gsysc:

- owner string less than 32 bytes
- owner string of 32 bytes
- owner string longer than 32 bytes
- no owner string
- owner string of a single null

Test Case Coverage for "SWB"

- CGSWBC - get SWB with:
  - owner string less than 32 bytes
  - owner string of 32 bytes
  - owner string longer than 32 bytes
  - no owner string
  - owner string of blanks
  - owner string of NULL
- CRSWBC - release all SWB pointers

Test Case Coverage for "Mixed"

- detac any data levels
- Utilize C++ operators:
  - new operator
  - delete operator
- Traditional Storage:
  - getcc attr - record id "OM"
  - getcc 381 block
  - getcc 1100 bytes, common
  - getcc 1055 block, blank fill
  - getcc 4095 block, common, blank fill
  - getcc 4095 block, protected, zero fill
  - relcc various levels (levtest)
  - getcc decb, attr - record id "OM"
  - getcc decb, 381 block
  - getcc decb, common, 1100 bytes
  - getcc decb, blank fill
  - getcc decb, common, blank fill
  - getcc decb, protected, zero fill
  - relcc all decbs
- SWB
- SHeap

- SHeap2
- relcc any data levels
- attac any blocks previously detached

Test Case Coverage for "Long Running Heap"

- QM5A will loop, acting as a long running ECB.
- Allocate 12 addresses using malloc, calloc, gsysc - various parameters
- If "last call" is NOT set, generate 6 random numbers and release only those addresses
- Next time through, allocate new addresses for any of the 12 addresses previously released
- If "last call" is NOT set, repeat random numbers and release
- If "last call" indicator is set, release anything outstanding and return to QM5A

Test Case Coverage for mallinfo - MALL

- Check values of ORDBLKS and KEEPCOST from mallinfo
- Check values of USMBLKS and FSMBLKS from mallinfo
- Malloc the maximum amount of heap storage allowed, using the FORDBLKS value, and check the UORDBLKS value
- Check values of SMBLKS and ARENA from mallinfo
- Check values of HBLKS and HBLKHD from mallinfo

Test Case Coverage for EBMAX - MAX

Normal test cases for tpf_ebmax:

- Use the system default maximum
- Successfully increase ECB maximum (for 31-bit ECB heap, 64-bit ECB heap, and ECB private area) by the tpf_ebmax() function.
- Increase ECB maximum (for 31-bit ECB heap, 64-bit ECB heap, and ECB private area) by the tpf_ebmax() function with a value larger than MAXVALUE. The ECB maximum values should stay the same.

Error test cases for tpf_ebmax:

- Allocate storage larger than the system default maximum values.
- Increase the maximum values, then allocate storage larger than the increased values.
- Increase the maximum values to the MAXVALUE, then allocate storage larger than the increased values.

Examples

The following example exercises the ECB heap support:

```
User:   ZTEST MEM5 API-EHEAP

System: ECBHeap: entering malloc_func for 31 bit  +
        ECBHeap: Successful malloc: addr 1D5F0100 1 bytes +
        ECBHeap: Successful free: address 1D5F0100 +
        ECBHeap: Successful malloc: addr 1D5F8400 4096 bytes +
        ECBHeap: Successful free: address 1D5F8400 +
        ECBHeap: Successful malloc: addr 1D60C008 65536 bytes +
        ECBHeap: Successful free: address 1D60C008 +
        ECBHeap: Successful malloc: addr 1D61C010 262144 bytes +
        ECBHeap: Successful free: address 1D61C010 +
        ECBHeap: Successful malloc: addr 1D61C010 100000 bytes +
        ECBHeap: Successful free: address 1D61C010 +
        ECBHeap: Successful calloc: addr 1D5F9408 10240 elements 1 bytes +
        ECBHeap: Successful malloc: addr 1D700000 19922936 bytes +
        ECBHeap: Successful malloc: addr 1D65C018 671712 bytes +
        ECBHeap: Successful malloc: addr 1D6346B8 162136 bytes +
        ECBHeap: Successful malloc: addr 1D61C010 100000 bytes +
        ECBHeap: Successful malloc: addr 1D60C008 65536 bytes +
        ECBHeap: Successful malloc: addr 1D600000 49152 bytes +
        ECBHeap: Successful free: address 1D700000 +
        ECBHeap: Successful free: address 1D65C018 +
        ECBHeap: Successful free: address 1D6346B8 +
        ECBHeap: Successful free: address 1D61C010 +
        ECBHeap: Successful free: address 1D60C008 +
        ECBHeap: Successful free: address 1D600000 +
        ECBHeap: Successful free: address 1D5F9408 +
        ECBHeap: entering malloc_func for 64 bit  +
        ECBHeap: Got sizes of buffs on ECB heap AVLn:
          HAVSZ1-64, HAVSZ2-256, HAVSZ3-1024, HAVSZ4-4096 +
        ECBHeap: Successful malloc: addr 480000000 63 bytes +
        ECBHeap: Successful free: address 480000000 +
        ECBHeap: Successful malloc: addr 480000100 255 bytes +
        ECBHeap: Successful free: address 480000100 +
        ECBHeap: Successful malloc: addr 480000500 1023 bytes +
        ECBHeap: Successful free: address 480000500 +
        ECBHeap: Successful malloc: addr 480001500 4095 bytes +
        ECBHeap: Successful free: address 480001500 +
        ECBHeap: Successful malloc: addr 480002508 4097 bytes +
        ECBHeap: Successful free: address 480002508 +
        ECBHeap: Successful malloc: addr 480002508 4097 bytes +
        ECBHeap: Tests tpf_eheap_tag( )  +
        ECBHeap: Tests error case: The first 8 bytes of the specified tag are not
         unique +
        ECBHeap: Successful malloc: addr 480003518 4097 bytes +
        ECBHeap: Successful free: address 480003518 +
        ECBHeap: Tests error case: The ECB heap buffer address specified is not
         valid +
        ECBHeap: Tests error case: The ECB heap buffer address specified already
         has a tag associated with it +
        ECBHeap: Testing tpf_eheap_locate(tagName)  +
        ECBHeap: Successful free: address 480002508 +
        ECBHeap: Successful malloc: addr 480001500 1 bytes +
        ECBHeap: Successful free: address 480001500 +
        ECBHeap: Successful malloc: addr 480001500 4096 bytes +
        ECBHeap: Successful free: address 480001500 +
        ECBHeap: Successful malloc: addr 480004528 65536 bytes +
        ECBHeap: Successful free: address 480004528 +
        ECBHeap: Successful malloc: addr 480014530 262144 bytes +
        ECBHeap: Successful free: address 480014530 +
        ECBHeap: Successful malloc: addr 480014530 100000 bytes +
        ECBHeap: Successful free: address 480014530 +
        ECBHeap: Successful malloc: addr 480100000 8388608 bytes +
        ECBHeap: Successful free: address 480100000 +
        ECBHeap: entering calloc_func for 31 bit  +
        ECBHeap: Successful calloc: addr 1D5F0040 1 elements 1 bytes +
        ECBHeap: Successful free: address 1D5F0040 +
        ECBHeap: Successful calloc: addr 1D5F0040 5 elements 5 bytes +
        ECBHeap: Successful free: address 1D5F0040 +
        ECBHeap: Successful calloc: addr 1D5F0100 1 elements 1024 bytes +
        ECBHeap: Successful free: address 1D5F0100 +
        ECBHeap: Successful calloc: addr 1D5F1100 1 elements 4096 bytes +
        ECBHeap: Successful free: address 1D5F1100 +
        ECBHeap: Successful calloc: addr 1D60C008 1 elements 65536 bytes +
        ECBHeap: Successful free: address 1D60C008 +
        ECBHeap: Successful calloc: addr 1D65C018 1 elements 262144 bytes +
        ECBHeap: Successful free: address 1D65C018 +
        ECBHeap: Successful calloc: addr 1D5F2108 1000 elements 8 bytes +
        ECBHeap: Successful free: address 1D5F2108 +
        ECBHeap: Successful calloc: addr 1D61C010 1 elements 100000 bytes +
        ECBHeap: Successful free: address 1D61C010 +
        ECBHeap: Successful calloc: addr 1D5F4050 10240 elements 1 bytes +
        ECBHeap: Successful calloc: addr 1D700000 19922936 elements 1 bytes +
        ECBHeap: Successful calloc: addr 1D69C020 409560 elements 1 bytes +
        ECBHeap: Successful calloc: addr 1D65C018 262144 elements 1 bytes +
        ECBHeap: Successful calloc: addr 1D6346B8 162136 elements 1 bytes +
        ECBHeap: Successful calloc: addr 1D61C010 100000 elements 1 bytes +
        ECBHeap: Successful calloc: addr 1D60C008 65536 elements 1 bytes +
        ECBHeap: Successful calloc: addr 1D600000 49152 elements 1 bytes +
        ECBHeap: Successful free: address 1D700000 +
        ECBHeap: Successful free: address 1D69C020 +
        ECBHeap: Successful free: address 1D65C018 +
        ECBHeap: Successful free: address 1D6346B8 +
        ECBHeap: Successful free: address 1D61C010 +
        ECBHeap: Successful free: address 1D60C008 +
        ECBHeap: Successful free: address 1D600000 +
        ECBHeap: Successful free: address 1D5F4050 +
        ECBHeap: entering calloc_func for 64 bit  +
        ECBHeap: Successful calloc: addr 480001500 1 elements 1 bytes +
        ECBHeap: Successful free: address 480001500 +
        ECBHeap: Successful calloc: addr 480001500 5 elements 5 bytes +
        ECBHeap: Successful free: address 480001500 +
        ECBHeap: Successful calloc: addr 480000500 1 elements 1024 bytes +
        ECBHeap: Successful free: address 480000500 +
        ECBHeap: Successful calloc: addr 480001500 1 elements 4096 bytes +
        ECBHeap: Successful free: address 480001500 +
        ECBHeap: Successful calloc: addr 480004528 1 elements 65536 bytes +
        ECBHeap: Successful free: address 480004528 +
        ECBHeap: Successful calloc: addr 480054538 1 elements 262144 bytes +
        ECBHeap: Successful free: address 480054538 +
        ECBHeap: Successful calloc: addr 480004528 1000 elements 8 bytes +
        ECBHeap: Successful free: address 480004528 +
        ECBHeap: Successful calloc: addr 480014530 1 elements 100000 bytes +
        ECBHeap: Successful free: address 480014530 +
        ECBHeap: Successful calloc: addr 480100000 1 elements 8388608 bytes +
        ECBHeap: Successful free: address 480100000 +
        ECBHeap: entering realloc_func for 31 bit  +
        ECBHeap: Successful malloc: addr 1D5F0100 150 bytes +
        ECBHeap: Successful realloc: addr 1D5F0040 1 bytes +
        ECBHeap: Successful free: address 1D5F0040 +
        ECBHeap: Successful malloc: addr 1D5F0040 10 bytes +
        ECBHeap: Successful realloc: addr 1D5F0200 1024 bytes +
        ECBHeap: Successful free: address 1D5F0200 +
        ECBHeap: Successful malloc: addr 1D5F1200 1000 bytes +
        ECBHeap: Successful realloc: addr 1D5F0100 100 bytes +
        ECBHeap: Successful free: address 1D5F0100 +
        ECBHeap: Successful calloc: addr 1D5F1600 2 elements 4096 bytes +
        ECBHeap: Successful realloc: addr 1D5F1600 4096 bytes +
        ECBHeap: Successful free: address 1D5F1600 +
        ECBHeap: Successful realloc: addr 1D5F1600 5000 bytes +
        ECBHeap: Successful free: address 1D5F1600 +
        ECBHeap: Successful calloc: addr 1D5F1200 1 elements 1000 bytes +
        ECBHeap: Successful realloc: addr 1D5F3608 65536 bytes +
        ECBHeap: Successful free: address 1D5F3608 +
        ECBHeap: Successful calloc: addr 1D5F1600 1 elements 5000 bytes +
        ECBHeap: Successful realloc: addr 1D603610 262144 bytes +
        ECBHeap: Successful free: address 1D603610 +
        ECBHeap: Successful calloc: addr 1D603610 1 elements 100000 bytes +
        ECBHeap: Successful realloc: addr 1D643618 200000 bytes +
        ECBHeap: Successful free: address 1D643618 +
        ECBHeap: Successful malloc: addr 1D5F3608 10240 bytes +
        ECBHeap: Successful realloc: addr 1D674360 20495512 bytes +
        ECBHeap: Successful free: address 1D674360 +
        ECBHeap: entering realloc_func for 64 bit  +
        ECBHeap: Successful malloc: addr 480000000 150 bytes +
        ECBHeap: Successful realloc: addr 480001500 1 bytes +
        ECBHeap: Successful free: address 480001500 +
        ECBHeap: Successful malloc: addr 480001500 10 bytes +
        ECBHeap: Successful realloc: addr 480000500 1024 bytes +
        ECBHeap: Successful free: address 480000500 +
        ECBHeap: Successful malloc: addr 480000100 1000 bytes +
        ECBHeap: Successful realloc: addr 480000000 100 bytes +
        ECBHeap: Successful free: address 480000000 +
        ECBHeap: Successful calloc: addr 480001500 2 elements 4096 bytes +
        ECBHeap: Successful realloc: addr 480001500 4096 bytes +
        ECBHeap: Successful free: address 480001500 +
        ECBHeap: Successful realloc: addr 480001500 5000 bytes +
        ECBHeap: Successful free: address 480001500 +
        ECBHeap: Successful calloc: addr 480000100 1 elements 1000 bytes +
        ECBHeap: Successful realloc: addr 480003508 65536 bytes +
        ECBHeap: Successful free: address 480003508 +
        ECBHeap: Successful calloc: addr 480001500 1 elements 5000 bytes +
        ECBHeap: Successful realloc: addr 480013510 262144 bytes +
        ECBHeap: Successful free: address 480013510 +
        ECBHeap: Successful calloc: addr 480013510 1 elements 100000 bytes +
        ECBHeap: Successful realloc: addr 480053518 200000 bytes +
        ECBHeap: Successful free: address 480053518 +
        ECBHeap: Successful malloc: addr 480084260 4194304 bytes +
        ECBHeap: Successful realloc: addr 480484268 4194305 bytes +
        ECBHeap: Successful free: address 480484268 +
        ECBHeap: entering realloc2_func mixing 31 and 64 bit +
        ECBHeap: Successful malloc: addr 1D5F0100 150 bytes +
        ECBHeap: Successful realloc: addr 480000500 1024 bytes +
        ECBHeap: Successful free: address 480000500 +
        ECBHeap: Successful malloc: addr 480000100 1000 bytes +
        ECBHeap: Successful realloc: addr 1D5F0100 100 bytes +
        ECBHeap: Successful free: address 1D5F0100 +
        ECBHeap: Successful calloc: addr 1D5F1200 1 elements 1000 bytes +
        ECBHeap: Successful realloc: addr 480001500 4096 bytes +
        ECBHeap: Successful free: address 480001500 +
        ECBHeap: Successful calloc: addr 480002508 1 elements 9000 bytes +
        ECBHeap: Successful realloc: addr 1D5F1600 5000 bytes +
        ECBHeap: Successful free: address 1D5F1600 +
        ECBHeap: entering free_func +
        ECBHeap: Successful free: address 0 +
        ECBHeap: Successful free: address 0 +
        ECBHeap: completed ECBHeap_func +
        MEM5 driver exited+
```

The following example exercises the system heap support:

```
User:   ZTEST MEM5 API-SHEAP

System: qm5a: entering systemheap_func +
        qm5a: Doing system heap test cases for TPF41 API +
        qm5a: End of Display +
        qm5a: Doing system heap test cases for zTPF API: 4K, 31BIT, no owner, non-unique token +
        qm5a: End of Display +
        qm5a: Doing system heap test cases for zTPF API: 4K, 31BIT, no owner, unique token +
        qm5a: End of Display +
        qm5a: Doing system heap test cases for zTPF API: 4K, 31BIT, owner, non-unique token +
        qm5a: End of Display +
        qm5a: Doing system heap test cases for zTPF API: 4K, 31BIT, owner, unique token+
        qm5a: End of Display +
        qm5a: Doing system heap test cases for zTPF API: 4K, 64BIT, no owner, non-unique token +
        qm5a: End of Display +
        qm5a: Doing system heap test cases for zTPF API: 4K, 64BIT, no owner, unique token +
        qm5a: End of Display +
        qm5a: Doing system heap test cases for zTPF API: 4K, 64BIT, owner, non-unique token +
        qm5a: End of Display +
        qm5a: Doing system heap test cases for zTPF API: 4K, 64BIT, owner of null, unique token +
        qm5a: End of Display +
        qm5a: Doing system heap test cases for zTPF API: 1MEG, 31BIT, no owner, non-unique token +
        qm5a: End of Display +
        qm5a: Doing system heap test cases for zTPF API: 1MEG, 31BIT, no owner, unique token +
        qm5a: End of Display +
        qm5a: Doing system heap test cases for zTPF API: 1MEG, 31BIT, owner, non-uniquetoken +
        qm5a: End of Display +
        qm5a: Doing system heap test cases for zTPF API: 1MEG, 31BIT, owner, unique token +
        qm5a: End of Display +
        qm5a: Doing system heap test cases for zTPF API: 1MEG, 64BIT, no owner, non-unique token +
        qm5a: End of Display +
        qm5a: Doing system heap test cases for zTPF API: 1MEG, 64BIT, no owner, unique token +
        qm5a: End of Display +
        qm5a: Doing system heap test cases for zTPF API: 1MEG, 64BIT, owner, non-unique token +
        qm5a: End of Display +
        qm5a: Doing system heap test cases for zTPF API: 1MEG, 64BIT, owner, unique token +
        qm5a: End of Display +
        qm5a: completed systemheap_func +
        MEM5 driver exited+
```

The following example exercises the SWB support:

```
User:   ZTEST MEM5 API-SWB

System: SWB: Successful CGSWBC: addr: AFA4C00 owner: owner less than 32 +
        SWB: Successful CRSWBC: addr: AFA4C00 +
        SWB: Successful CGSWBC: addr: AFA3000 owner: 123456789012345678901234567890123+
        SWB: Successful CRSWBC: addr: AFA3000 +
        SWB: Successful CGSWBC: addr: AFA5000 owner: this owner string is equal to 32  +
        SWB: Successful CRSWBC: addr: AFA5000 +
        SWB: Successful CGSWBC: addr: AFA6400 owner:                                   +
        SWB: Successful CRSWBC: addr: AFA6400 +
        SWB: Successful CGSWBC: addr: AFA9000 owner: 0                                 +
        SWB: Successful CRSWBC: addr: AFA9000 +
        SWB: completed SWB_func +
        MEM5 driver exited+
```

The following example exercises the long running heap support:

```
User:   ZTEST MEM5 API-LONG

System: Long Running Heap: Test case completed. Releasing all storage. +
        Long Running Heap: completed LongRunningHeap_func +
        MEM5 driver exited+
```

The following example exercises the mallinfo support:

```
User:   ZTEST MEM5 API-MALL

System: MallInfo: Entering MallInfo_func +
        MallInfo: case 1: testing ordblks and keepcost +
        MallInfo: Successful calloc: addr 1D5F8400 10240 elements 1 bytes +
        MallInfo: Successful malloc: addr 1D600000 20971512 bytes +
        MallInfo: case 1 successful: ordblks = 20, keepcost = 20 +
        MallInfo: Successful free: address 1D600000 +
        MallInfo: Successful free: address 1D5F8400 +
        MallInfo: case 2: testing usmblks and fsmblks +
        MallInfo: Successful malloc: addr 1D5FAC08 21488 bytes +
        MallInfo: Successful free: address 1D5FAC08 +
        MallInfo: case 3: testing fordblks and uordblks +
        MallInfo: Successful malloc: addr 1D600000 20971512 bytes +
        MallInfo: Successful free: address 1D600000 +
        MallInfo: case 4: testing arena and smbblks +
        MallInfo: case 5: testing hblks and hblkhd +
        MallInfo: completed MallInfo_func +
        MEM5 driver exited+
```

The following example exercises the EBMAX support:

```
User:   ZTEST MEM5 API-MAX

System: ECB Max: entering ECBMax_func +
        ECB Max: Running Normal Cases for ECBMax_func +
        ECB Max: Successful malloc: addr 1D600000 20971512 bytes +
        ECB Max: Successful malloc: addr 480000000 8388608 bytes +
        ECB Max: Successful free: address 1D600000 +
        ECB Max: Successful free: address 480000000 +
        ECB Max: Creating child ECB to run normal ECB Private Area case +
        ECB Max: Successful malloc: addr 1D600000 20971512 bytes +
        ECB Max: Successful malloc: addr 480000000 2097152 bytes +
        ECB Max: Successful free: address 1D600000 +
        ECB Max: Successful free: address 480000000 +
        ECB Max: Completed Normal Cases for ECBMax_func +
        ECB Max: completed ECBMax_func +
        MEM5 driver exited+
```

References

For more information about reading syntax diagrams, also referred to as railroad diagrams, see *Accessibility information* in the TPF Product Information Center.