# Main Page

* [CFS Memory Manager Introduction](#AAAAAAAAKN)
* [CFS Memory Manager Overview](#AAAAAAAAKO)
* [CFS Memory Manager Operation](#AAAAAAAAKU)
* [CFS Memory Manager Requirements](#AAAAAAAAKV)
* [CFS Memory Manager Deployment Guide](#AAAAAAAAKQ)
* [CFS Memory Manager Commands](#AAAAAAAAKX)
* [CFS Memory Manager Telemetry](#AAAAAAAAKY)
* [CFS Memory Manager Table Definitions](#AAAAAAAAKZ)
* [CFS Memory Manager Configuration Parameters](#AAAAAAAAKW)
* [CFS Memory Manager Operational Constraints](#AAAAAAAALO)
* [CFS Memory Manager Frequently Asked Questions](#AAAAAAAAMT)
* [CFS Memory Manager Event Message Reference](#AAAAAAAAEF)

# CFS Memory Manager Introduction

#### Scope

This document provides a complete specification for the commands and telemetry associated with the CFS Memory Manager (MM) application software. The document is intended primarily for users of the software (operations personal, test engineers, and maintenance personnel). The last section of the document, the deployment guide section, is intended for mission developers when deploying and configuring the MM application software for a mission flight software build environment.

#### Applicable Documents

|  |  |
| --- | --- |
| **Document ID** | **Document Title** |
| TBD | CFS Memory Manager Application Requirements Document |
| TBD | CFS Memory Manager Heritage Analysis Document |
| TBD | CFS Memory Manager Design Document |

#### Acronyms

|  |  |
| --- | --- |
| **Acronym** | **Description** |
| DS | CFS Data Storage |
| API | Application Programming Interface |
| ATP | Absolute Time Processor |
| ATS | Absolute Time tagged command Sequence |
| CCSDS | Consultative Committee for Space Data Systems |
| C&DH | Command and Data Handling |
| CFDP | CCSDS File Delivery Protocol |
| CFE | Core Flight Executive |
| CFS | Core Flight System |
| CI | Command Ingest |
| Cmd | Command |
| CPU | Central Processing Unit |
| EDAC | Error Detection and Correction |
| FDS | Flight Data System |
| FM | File Manager |
| FSW | Flight Software |
| GN&C | Guidance Navigation & Control |
| GSFC | Goddard Space Flight Center |
| HK | Housekeeping |
| HW, H/W | Hardware |
| ICD | Interface Control Document |
| ISR | Interrupt Service Routine |
| OS | Operating System |
| OSAL | Operating System Abstraction Layer |
| Pkts | Packets |
| PSP | Platform Support Package |
| RAM | Random-Access Memory |
| RTOS | Real Time Operating System |
| RTP | Relative Time Processor |
| RTS | Relative Time tagged command Sequence |
| SB | Software Bus Service |
| SBC | Single Board Computer |
| SC | Stored Commands Application |
| SW, S/W | Software |
| TBD | To Be Determined |
| TBL | Table |
| TDM | Time Data Multiplex |
| TLM | Telemetry |
| UTC | Universal time code |

# CFS Memory Manager Overview

The Memory Manager (MM) component of the Core Flight System (CFS) is responsible for the loading and dumping of flight system memory. MM is basically the operator interface for the PSP (Platform Support Package) and OSAL (Operating System Abstraction Layer) memory manipulation functions. Memory Manager provides the ability to load and dump memory via commands as well as from files. If the operating system supports symbolic addressing, Memory Manager allows specifying the memory address using a symbolic address.

Since MM makes use of the PSP and OSAL when interfacing to memory, it relies on PSP and OSAL routines to access processor memory as well as memory that is not directly accessible (i.e. requires address translation). MM does no memory manipulation that bypasses the OSAL. Address checking is performed using the PSP and any addresses specified outside of defined legal address ranges will be considered invalid and an error event will be issued.

MM performs data transfers between memory and files, but does not handle file transfers to (or from) the ground. That function must be done with a file transfer application such as the CFS CFDP application. MM also does not have responsibility for file management operations or directory manipulations. That function is allocated to the CFS FM (File Manager) application.

MM is a command driven, single threaded application. It will pend indefinitely on the software bus message queue until commands are recieved and then process them sequentially in the order they arrived. Memory Manager segments loads and dumps in order to prevent CPU hogging. Despite this, careful consideration should be made when loading and dumping large amounts of memory. See [CFS Memory Manager Deployment Guide](#AAAAAAAAKQ).

For additional details see the following sections:

* [Memory Types](#AAAAAAAAKR)
* [File Format](#AAAAAAAAKS)
* [Symbolic Address Arguments](#AAAAAAAAKT)

## Memory Types

There are five different memory types that can be specified in MM commands. These are:

MM\_RAM Generic term for RAM including DRAM, and SRAM. Memory mapped I/O that is byte addressable and requires no special code to support is accessed with this memory type.

MM\_EEPROM Generic term used for non-volatile memory including EEPROM, Flash, PROM, etc. MM places no address or data alignment restrictions on this memory type. Any special alignment requirements imposed by the hardware are handled in the OSAL code for the target platform in question. MM provides commands to enable and disable EEPROM write protection via a PSP interface.

MM\_MEM32 Addressable memory that must be read from and written to in 32 bit double words. Inclusion of this type is controlled by a conditional compile switch. For this memory type, MM will use the #CFE\_PSP\_MemRead32 and #CFE\_PSP\_MemWrite32 functions instead of the more generic #CFE\_PSP\_MemCpy. Address and data size parameters for this memory type must be 32 bit aligned or an error event will be issued and the current operation aborted.

MM\_MEM16 Addressable memory that must be read from and written to in 16 bit words. Inclusion of this type is controlled by a conditional compile switch. For this memory type, MM will use the #CFE\_PSP\_MemRead16 and #CFE\_PSP\_MemWrite16 OSAL functions instead of the more generic #CFE\_PSP\_MemCpy. Address and data size parameters for this memory type must be 16 bit aligned or an error event will be issued and the current operation aborted.

MM\_MEM8 Addressable memory that must be read from and written to in 8 bit bytes. Inclusion of this type is controlled by a conditional compile switch. For this memory type, MM will use the #CFE\_PSP\_MemRead8 and #CFE\_PSP\_MemWrite8 OSAL functions instead of the more generic #CFE\_PSP\_MemCpy.

## File Format

The MM application uses the same format for both load and dump files and does not distinguish between the two. This allows the contents of a memory dump file to be loaded back to the same location in memory if desired, providing a memory save and restore capability to the ground operators.

Each dump (and load) file has three parts:

A file primary header. This is the standard cFE file header whose structure is defined by the #CFE\_FS\_Header\_t data type.

The MM secondary header defined by the [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) data type.

A variable size block of binary data that holds the memory contents dumped or the data to be loaded.

On some operating systems, MM also provides the ability to dump the symbol table to a file. The resulting symbol table file does not include the standard cFE file header.

## Symbolic Address Arguments

The MM application supports the use of symbolic names instead of absolute memory addresses. This is especially useful in environments where dynamic loading is used. Addresses are specified in two parts, a character string that holds the symbol name, and an optional numerical offset as defined in the #CFS\_SymAddr\_t data type.

This data structure is used for all address arguments in ground commands and load files even if symbol table operations are not implemented on the target platform. This makes command and file formats consistent across missions and enhances the portability of ground system scripts and database definitions.

Symbol processing is as follows:

If the symbol name string is not a NUL (empty, or zero length) string, an attempt is made to resolve the name to a physical hardware address through the OSAL. If the resolution succeeds, the offset argument is added to the address and the resulting fully resolved address is used for the memory operation requested. If the symbol name cannot be resolved by the OSAL (either the symbol name is invalid or no symbol table is available), MM will issue an error event and abort the current operation.

If the symbol name string is a NUL string, no resolution through the OSAL takes place and the offset is treated as an absolute address. This is how address arguments would be specified when there is no symbol support on the target platform.

It should be noted that all addresses reported by MM in housekeeping telemetry and dump file headers are always the fully resolved addresses, even if a symbol or combination symbol/offset were used as specifiers when a memory operation was initiated.

#### Dumping the Symbol Table

A ground command is available in the MM app that allows the user to dump the OS Symbol Table to a file. The MM app calls the OSAL symbol table dump utility, which performs the operation. The dump file name is specified in the ground command and is limited to #OS\_MAX\_PATH\_LEN - 1 characters in size. Once the dump is completed, the file may be downloaded.

# CFS Memory Manager Operation

#### Initialization

Because MM is bascially an interface to the PSP and OSAL memory utilities, the app only performs basic initialization steps (e.g., registering the app, subscribing to housekeeping and ground commands, initializing counters and housekeeping data). MM does not maintain any tables or critical data storage (CDS) areas, so no actions are taken during initialization.

Once initialization is complete, MM will pend indefinitely for a command (either ground or housekeeping) and wake up to process the received command.

# CFS Memory Manager Requirements

For detailed MM requirements, please see the "Core Flight System (CFS) Memory Manager Application Requirements" Document number: 582-2007-031.

# CFS Memory Manager Deployment Guide

To integrate the MM application into the cFE, follow the CFS Deployment Guide.

Follow the general guidelines below for platform deployment of the Memory Manager Application.

Since the MM application uses files for dumps and loads, directories and file volumes need to be setup and initialized correctly through the cFE for these operations to work properly. MM will use the full path specified in a command message (up to the max specified by #OS\_MAX\_PATH\_LEN) without modification for file operations.

MM relies on the OSAL for validation of address arguments. It uses the function #CFE\_PSP\_MemValidateRange to check address and data size parameters for dump and load operations. The memory map used by #CFE\_PSP\_MemValidateRange needs to be setup correctly for the platform, otherwise these parameter checks may fail unexpectedly.

Code to support the MM\_MEM32, MM\_MEM16, and MM\_MEM8 memory types (see [Memory Types](#AAAAAAAAKR)) is conditionally compiled based upon configuration parameter settings (see [CFS Memory Manager Configuration Parameters](#AAAAAAAAKW)). If support for these types is not needed, excluding them can reduce the MM code footprint.

The maximum amount of data that can be dumped to (or loaded from) a file can also be changed by adjusting configuration parameter settings (see [CFS Memory Manager Configuration Parameters](#AAAAAAAAKW)). Since MM processes commands and housekeeping requests serially from a single command pipe, setting these limits to large values may cause late responses to housekeeping requests. If a mission requires large dumps or loads, it may be better to split the operation so it uses several smaller files instead of one very large one.

Only one message ID must be included in the CFE Scheduler Table: [MM\_SEND\_HK\_MID](#AAAAAAAAAO) is sent out at the housekeeping request interval.

The Memory Manager app generates telemetry when it receives the housekeeping request. Its telemetry message ID is [MM\_HK\_TLM\_MID](#AAAAAAAAAP).

The cFE ES app uses five performance IDs to keep track of the performance of the MM app: [MM\_APPMAIN\_PERF\_ID](#AAAAAAAAAH) is used to track overall app performance.

[MM\_SEGBREAK\_PERF\_ID](#AAAAAAAAAI) is used to track performance during segment breaks, which prevent CPU hogging during load-from-file, dump-to-file, or fill operations.

[MM\_EEPROM\_POKE\_PERF\_ID](#AAAAAAAAAJ) is used to track performance during poke operations.

[MM\_EEPROM\_FILELOAD\_PERF\_ID](#AAAAAAAAAK) is used to track performance during file load operations.

[MM\_EEPROM\_FILL\_PERF\_ID](#AAAAAAAAAL) is used to track performance during fill operations.

The mission configuration file [mm\_mission\_cfg.h](#AAAAAAAAAB) contains parameters that are used across all platforms on a mission. The defined parameters and their default values are:

[MM\_LOAD\_WID\_CRC\_TYPE](#AAAAAAAAAC) specifies the CRC algorithm that MM will request when calling the Executive Services CRC utility during a "memory load with interrupts disabled" ([MM\_LOAD\_MEM\_WID\_CC](#AAAAAAAAAF)) command. The default value is the ES default #CFE\_ES\_DEFAULT\_CRC.

##### Note:

Only the default ES CRC algorithm (16-bit CRC) is currently available in the ES CRC utility. Although other definitions (8-bit and 32-bit) exist, those algorithms have not been implemented, and setting the MM type to those values (or to other, undefined values) could lead to unpredictable results.

[MM\_LOAD\_FILE\_CRC\_TYPE](#AAAAAAAAAD) specifies the CRC algorithm that MM will request when calling the Executive Services CRC utility during a memory load from file. The default value is the ES default #CFE\_ES\_DEFAULT\_CRC.

##### Note:

Only the default ES CRC algorithm (16-bit CRC) is currently available in the ES CRC utility. Although other definitions (8-bit and 32-bit) exist, those algorithms have not been implemented, and setting the MM type to those values (or to other, undefined values) could lead to unpredictable results.

[MM\_DUMP\_FILE\_CRC\_TYPE](#AAAAAAAAAE) specifies the CRC algorithm that MM will request when calling the Executive Services CRC utility during a memory dump to file. The default value is the ES default #CFE\_ES\_DEFAULT\_CRC.

##### Note:

Only the default ES CRC algorithm (16-bit CRC) is currently available in the ES CRC utility. Although other definitions (8-bit and 32-bit) exist, those algorithms have not been implemented, and setting the MM type to those values (or to other, undefined values) could lead to unpredictable results.

The platform configuration file [mm\_platform\_cfg.h](#AAAAAAAAAQ) contains parameters that can be adjusted to specific platforms. The defined parameters (and their default values) are:

[MM\_MAX\_LOAD\_FILE\_DATA\_RAM](#AAAAAAAAAT) defines the maximum number of bytes that can be loaded into RAM from a single load file. The default value is 1048576. This value should not be set so large that it prevents MM from responding to housekeeping requests during a load.

[MM\_MAX\_LOAD\_FILE\_DATA\_EEPROM](#AAAAAAAAAU) defines the maximum number of bytes that can be loaded into EEPROM from a single load file. The default value is 1048576. This value should not be set so large that it prevents MM from responding to housekeeping requests during a load.

#MM\_MAX\_UNINTERRUPTABLE\_DATA defines the maximum number of bytes that can be loaded when interrupts are disable (see [MM\_LOAD\_MEM\_WID\_CC](#AAAAAAAAAF)). The default value is 200. This value should not be set so large that interrupts are disabled for an unacceptable amount of time, and it should also be small enough to avoid packet segmentation for the command protocol being used..

[MM\_MAX\_LOAD\_DATA\_SEG](#AAAAAAAAAW) defines the number of bytes MM can load per cycle to prevent CPU hogging. The default value is 200. Setting this value too high will increase MM CPU utilization during loads and will decrease the amount of execution time available for other tasks.

[MM\_MAX\_DUMP\_FILE\_DATA\_RAM](#AAAAAAAAAX) defines the maximum number of bytes that can be dumped from RAM into a single dump file. The default value is 1048576. This value should not be set so large that it prevents MM from responding to housekeeping requests during a dump.

[MM\_MAX\_DUMP\_FILE\_DATA\_EEPROM](#AAAAAAAAAY) defines the maximum number of bytes that can be dumped from EEPROM into a single dump file. The default value is 1048576. This value should not be set so large that it prevents MM from responding to housekeeping requests during a dump.

[MM\_MAX\_DUMP\_FILE\_DATA\_SYMTBL](#AAAAAAAAAZ) defines the maximum number of bytes that can be dumped from the symbol table into a single dump file. The default value is 131072. This value is passed to the OSAL for dump processing, and should not be set so large that OSAL rejects the dump request.

[MM\_MAX\_DUMP\_DATA\_SEG](#AAAAAAAABA) defines the number of bytes MM can dump per cycle to prevent CPU hogging. The default value is 200. Setting this value too high will increase MM CPU utilization during dumps and will decrease the amount of execution time available for other tasks.

[MM\_MAX\_FILL\_DATA\_RAM](#AAAAAAAABB) defines the number of bytes that can be loaded into RAM with a single memory fill command. The default value is 1048576. This value should not be set so large that it prevents MM from responding to housekeeping requests during a memory fill.

[MM\_MAX\_FILL\_DATA\_EEPROM](#AAAAAAAABC) defines the number of bytes that can be loaded into EEPROM with a single memory fill command. The default value is 131072. This value should not be set so large that it prevents MM from responding to housekeeping requests during a memory fill.

[MM\_MAX\_FILL\_DATA\_SEG](#AAAAAAAABD) defines the number of bytes MM can fill per cycle to prevent CPU hogging. The default value is 200. Setting this value too high will increase MM CPU utilization during memory fills and will decrease the amount of execution time available for other tasks.

[MM\_OPT\_CODE\_MEM32\_MEMTYPE](#AAAAAAAABE) defines whether code for [MM\_MEM32](#AAAAAAAAIJ) memory operations should be included in the MM app. The default value is TRUE, which includes the code. The value must be either TRUE or FALSE.

[MM\_MAX\_LOAD\_FILE\_DATA\_MEM32](#AAAAAAAABF) defines the maximum number of bytes that can be loaded into the optional [MM\_MEM32](#AAAAAAAAIJ) memory type from a single load file. The default value is 1048576. This value must be longword aligned, and should not be set so large that it prevents MM from responding to housekeeping requests during a load.

[MM\_MAX\_DUMP\_FILE\_DATA\_MEM32](#AAAAAAAABG) defines the maximum number of bytes that can be dumped from the optional [MM\_MEM32](#AAAAAAAAIJ) memory type into a single load file. The default value is 1048576. This value must be longword aligned, and should not be set so large that it prevents MM from responding to housekeeping requests during a load.

[MM\_MAX\_FILL\_DATA\_MEM32](#AAAAAAAABH) defines the maximum number of bytes that can be loaded into the optional [MM\_MEM32](#AAAAAAAAIJ) memory type from a single memory fill command. The default value is 1048576. This value must be longword aligned, and should not be set so large that it prevents MM from responding to housekeeping requests during a memory fill.

[MM\_OPT\_CODE\_MEM16\_MEMTYPE](#AAAAAAAABI) defines whether code for [MM\_MEM16](#AAAAAAAAIE) memory operations should be included in the MM app. The default value is TRUE, which includes the code. The value must be either TRUE or FALSE.

[MM\_MAX\_LOAD\_FILE\_DATA\_MEM16](#AAAAAAAABJ) defines the maximum number of bytes that can be loaded into the optional [MM\_MEM16](#AAAAAAAAIE) memory type from a single load file. The default value is 1048576. This value must be word aligned, and should not be set so large that it prevents MM from responding to housekeeping requests during a load.

[MM\_MAX\_DUMP\_FILE\_DATA\_MEM16](#AAAAAAAABK) defines the maximum number of bytes that can be dumped from the optional [MM\_MEM16](#AAAAAAAAIE) memory type into a single load file. The default value is 1048576. This value must be word aligned, and should not be set so large that it prevents MM from responding to housekeeping requests during a load.

[MM\_MAX\_FILL\_DATA\_MEM16](#AAAAAAAABL) defines the maximum number of bytes that can be loaded into the optional [MM\_MEM16](#AAAAAAAAIE) memory type from a single memory fill command. The default value is 1048576. This value must be word aligned, and should not be set so large that it prevents MM from responding to housekeeping requests during a memory fill.

[MM\_OPT\_CODE\_MEM8\_MEMTYPE](#AAAAAAAABM) defines whether code for [MM\_MEM8](#AAAAAAAAIP) memory operations should be included in the MM app. The default value is TRUE, which includes the code. The value must be either TRUE or FALSE.

[MM\_MAX\_LOAD\_FILE\_DATA\_MEM8](#AAAAAAAABN) defines the maximum number of bytes that can be loaded into the optional [MM\_MEM8](#AAAAAAAAIP) memory type from a single load file. The default value is 1048576. This value should not be set so large that it prevents MM from responding to housekeeping requests during a load.

[MM\_MAX\_DUMP\_FILE\_DATA\_MEM8](#AAAAAAAABO) defines the maximum number of bytes that can be dumped from the optional [MM\_MEM8](#AAAAAAAAIP) memory type into a single load file. The default value is 1048576. This value should not be set so large that it prevents MM from responding to housekeeping requests during a load.

[MM\_MAX\_FILL\_DATA\_MEM8](#AAAAAAAABP) defines the maximum number of bytes that can be loaded into the optional [MM\_MEM8](#AAAAAAAAIP) memory type from a single memory fill command. The default value is 1048576. This value should not be set so large that it prevents MM from responding to housekeeping requests during a memory fill.

[MM\_PROCESSOR\_CYCLE](#AAAAAAAABQ) defines the number of milliseconds MM should delay between segments for dump, load and fill operations. The default value is 0, which means that MM will give up what's left of the current timeslice. The value should not be set so large that it prevents MM from responding to housekeeping requests. Additionally, it should not be set so small (non-zero) that it prevents other applications from running.

[MM\_MISSION\_REV](#AAAAAAAABR) defines the mission-level revision number. The default value is 0. This value is appended to the major version, minor version, and revision numbers. It should be an integer value greater than or equal to 0.

# CFS Memory Manager Commands

The following is a list of commands that are processed by the CFS Memory Manager Application.

##### Global [MM\_DISABLE\_EEPROM\_WRITE\_CC](#AAAAAAAACF)

EEPROM Write Disable

##### Global [MM\_DUMP\_IN\_EVENT\_CC](#AAAAAAAADT)

Dump In Event Message

##### Global [MM\_DUMP\_MEM\_TO\_FILE\_CC](#AAAAAAAADV)

Memory Dump To File

##### Global [MM\_ENABLE\_EEPROM\_WRITE\_CC](#AAAAAAAACG)

EEPROM Write Enable

##### Global [MM\_FILL\_MEM\_CC](#AAAAAAAAHL)

Memory Fill

##### Global [MM\_LOAD\_MEM\_FROM\_FILE\_CC](#AAAAAAAAHN)

Memory Load From File

##### Global [MM\_LOAD\_MEM\_WID\_CC](#AAAAAAAAAF)

Memory Load With Interrupts Disabled

##### Global [MM\_LOOKUP\_SYM\_CC](#AAAAAAAACH)

Symbol Table Lookup

##### Global [MM\_NOOP\_CC](#AAAAAAAACI)

Noop

##### Global [MM\_PEEK\_CC](#AAAAAAAADX)

Memory Peek

##### Global [MM\_POKE\_CC](#AAAAAAAAHO)

Memory Poke

##### Global [MM\_RESET\_CC](#AAAAAAAACJ)

Reset Counters

##### Global [MM\_SYMTBL\_TO\_FILE\_CC](#AAAAAAAACK)

Save Symbol Table To File

# CFS Memory Manager Telemetry

The following are telemetry packets generated by the CFS Memory Manager Application.

##### 

##### Class [MM\_HkPacket\_t](#AAAAAAAAIY)

Housekeeping Packet Structure

# CFS Memory Manager Table Definitions

The CFS Memory Manager application does not utilize or manage any software tables.

# CFS Memory Manager Configuration Parameters

The following are configuration parameters used to configure the CFS Memory Manager Application either for each platform or for a mission as a whole.

##### Global [MM\_CFE\_HDR\_DESCRIPTION](#AAAAAAAAAS)

Memory Management File – cFE file header description

##### Global [MM\_CFE\_HDR\_SUBTYPE](#AAAAAAAAAR)

Memory Managment File – cFE file header sub-type

##### Global [MM\_DUMP\_FILE\_CRC\_TYPE](#AAAAAAAAAE)

CRC type for dump files

##### Global [MM\_LOAD\_FILE\_CRC\_TYPE](#AAAAAAAAAD)

CRC type for load files

##### Global [MM\_LOAD\_WID\_CRC\_TYPE](#AAAAAAAAAC)

CRC type for interrupts disabled loads

##### Global [MM\_MAX\_DUMP\_DATA\_SEG](#AAAAAAAABA)

Maximum number of bytes per dump data segment

##### Global [MM\_MAX\_DUMP\_FILE\_DATA\_EEPROM](#AAAAAAAAAY)

Maximum number of bytes for a file dump from EEPROM memory

##### Global [MM\_MAX\_DUMP\_FILE\_DATA\_MEM16](#AAAAAAAABK)

Maximum number of bytes for a file dump from MEM16 memory

##### Global [MM\_MAX\_DUMP\_FILE\_DATA\_MEM32](#AAAAAAAABG)

Maximum number of bytes for a file dump from MEM32 memory

##### Global [MM\_MAX\_DUMP\_FILE\_DATA\_MEM8](#AAAAAAAABO)

Maximum number of bytes for a file dump from MEM8 memory

##### Global [MM\_MAX\_DUMP\_FILE\_DATA\_RAM](#AAAAAAAAAX)

Maximum number of bytes for a file dump from RAM memory

##### Global [MM\_MAX\_DUMP\_FILE\_DATA\_SYMTBL](#AAAAAAAAAZ)

Maximum number of bytes for a symbol table file dump

##### Global [MM\_MAX\_FILL\_DATA\_EEPROM](#AAAAAAAABC)

Maximum number of bytes for a fill to EEPROM memory

##### Global [MM\_MAX\_FILL\_DATA\_MEM16](#AAAAAAAABL)

Maximum number of bytes for a fill to MEM16 memory

##### Global [MM\_MAX\_FILL\_DATA\_MEM32](#AAAAAAAABH)

Maximum number of bytes for a fill to MEM32 memory

##### Global [MM\_MAX\_FILL\_DATA\_MEM8](#AAAAAAAABP)

Maximum number of bytes for a fill to MEM8 memory

##### Global [MM\_MAX\_FILL\_DATA\_RAM](#AAAAAAAABB)

Maximum number of bytes for a fill to RAM memory

##### Global [MM\_MAX\_FILL\_DATA\_SEG](#AAAAAAAABD)

Maximum number of bytes per fill data segment

##### Global [MM\_MAX\_LOAD\_DATA\_SEG](#AAAAAAAAAW)

Maximum number of bytes per load data segment

##### Global [MM\_MAX\_LOAD\_FILE\_DATA\_EEPROM](#AAAAAAAAAU)

Maximum number of bytes for a file load to EEPROM memory

##### Global [MM\_MAX\_LOAD\_FILE\_DATA\_MEM16](#AAAAAAAABJ)

Maximum number of bytes for a file load to MEM16 memory

##### Global [MM\_MAX\_LOAD\_FILE\_DATA\_MEM32](#AAAAAAAABF)

Maximum number of bytes for a file load to MEM32 memory

##### Global [MM\_MAX\_LOAD\_FILE\_DATA\_MEM8](#AAAAAAAABN)

Maximum number of bytes for a file load to MEM8 memory

##### Global [MM\_MAX\_LOAD\_FILE\_DATA\_RAM](#AAAAAAAAAT)

Maximum number of bytes for a file load to RAM memory

##### Global [MM\_MAX\_UNINTERRUPTIBLE\_DATA](#AAAAAAAAAV)

Maximum number of bytes for an uninterruptable load

##### Global [MM\_MISSION\_REV](#AAAAAAAABR)

Mission specific version number for MM application

##### Global [MM\_OPT\_CODE\_MEM16\_MEMTYPE](#AAAAAAAABI)

Optional MEM16 compile switch

##### Global [MM\_OPT\_CODE\_MEM32\_MEMTYPE](#AAAAAAAABE)

Optional MEM32 compile switch

##### Global [MM\_OPT\_CODE\_MEM8\_MEMTYPE](#AAAAAAAABM)

Optional MEM8 compile switch

##### Global [MM\_PROCESSOR\_CYCLE](#AAAAAAAABQ)

Segment break processor delay

# CFS Memory Manager Operational Constraints

When loading memory from a file, it is the responsibility of the operators to ensure that newly loaded symbol data fits within old symbol space. There is currently no ability in MM to unload a symbol and load it back or dynamically check for dependencies for the new code.

The Dump Symbol Table to File command does not work on the Linux operating system.

# CFS Memory Manager Frequently Asked Questions

**(Q) How can I tell if a bank of EEPROM is write-enabled or write-disabled?**

*Currently, the write enable/disable status is maintained internally by the PSP. There is no PSP utility available to return the write status of a bank of EEPROM. The only way to know if a bank of EEPROM is disabled is if the PSP returns an error status back to MM during a write operation.*

*Future releases of MM will include a ground command to return the write enable status of a selected bank.*

**(Q) Can I write to the same area of memory that is currently being dumped?**

*Yes. The PSP does not prevent an area of memory that is being dumped from being written to at the same. The only constraint is that the two actions cannot access the same file (if the load-from-file and dump-to-file actions are being used). You should be careful to make sure that any memory writes are completed before dumping the same area of memory, and vice versa.*

**(Q) What's the difference between the default** [**MM\_RAM**](#AAAAAAAAJE) **memory type and the** [**MM\_MEM8**](#AAAAAAAAIP) **memory type? If I need to read or write an odd number of bytes why couldn't I use the default commands?**

*The load and dump operations on* [*MM\_RAM*](#AAAAAAAAJE) *can read/write an odd number of bytes, but the low-level implementation (the utilities called by the operations) differs from those used on* [*MM\_MEM8*](#AAAAAAAAIP) *memory type operations.* [*MM\_MEM8*](#AAAAAAAAIP) *load/dump operations use the #CFE\_PSP\_MemRead8 and #CFE\_PSP\_MemWrite8 utilities, which do byte-by-byte reads and writes. The* [*MM\_RAM*](#AAAAAAAAJE) *load/dump operations use the C-library memcpy function, which can be dependent on the compiler you use for your mission. This may be acceptable, but* [*MM\_MEM8*](#AAAAAAAAIP) *is available to guarantee byte-by-byte reads and writes.*

**(Q) I don't need a special 32-bit/16-bit/8-bit load and dump capability for my mission. Do I have to leave this code in the application?**

*No.* [*MM\_MEM8*](#AAAAAAAAIP)*,* [*MM\_MEM16*](#AAAAAAAAIE)*, and* [*MM\_MEM32*](#AAAAAAAAIJ) *memory types are conditionally compiled in the application through the switches* [*MM\_OPT\_CODE\_MEM8\_MEMTYPE*](#AAAAAAAABM)*,* [*MM\_OPT\_CODE\_MEM16\_MEMTYPE*](#AAAAAAAABI)*, and* [*MM\_OPT\_CODE\_MEM32\_MEMTYPE*](#AAAAAAAABE)*, respectively. Setting any of these switches to FALSE in the* [*mm\_platform\_cfg.h*](#AAAAAAAAAQ) *configuration file will exclude that code from your application.*

*If your ground system includes commands for these operations, MM will return an "invalid ground command" error when you send a command that uses the excluded code. Those commands should be removed from your operational ground system to avoid inadvertant transmission.*

# MM Telemetry Mnemonic Cross Reference

##### Global [MM\_HkPacket\_t::Address](#AAAAAAAAMV)

##### Global [MM\_HkPacket\_t::BytesProcessed](#AAAAAAAAMX)

##### Global [MM\_HkPacket\_t::CmdCounter](#AAAAAAAAMZ)

##### Global [MM\_HkPacket\_t::DataValue](#AAAAAAAANB)

##### Global [MM\_HkPacket\_t::ErrCounter](#AAAAAAAAND)

##### Global [MM\_HkPacket\_t::FileName](#AAAAAAAANF) [OS\_MAX\_PATH\_LEN]

##### Global [MM\_HkPacket\_t::LastAction](#AAAAAAAANH)

##### Global [MM\_HkPacket\_t::MemType](#AAAAAAAANJ)

# MM Event Message Cross Reference

##### Global [MM\_ALIGN16\_ERR\_EID](#AAAAAAAAFD)

'Data and address not 16 bit aligned: Addr = 0x%08X Size = %d'

##### Global [MM\_ALIGN32\_ERR\_EID](#AAAAAAAAFC)

'Data and address not 32 bit aligned: Addr = 0x%08X Size = %d'

##### Global [MM\_CC1\_ERR\_EID](#AAAAAAAAEW)

'Invalid ground command code: ID = 0x%X, CC = %d'

##### Global [MM\_CFE\_FS\_READHDR\_ERR\_EID](#AAAAAAAAFV)

'CFE\_FS\_ReadHeader error received: RC = 0x%08X Expected = %d File = '%s''

##### Global [MM\_CFE\_FS\_WRITEHDR\_ERR\_EID](#AAAAAAAAFW)

'CFE\_FS\_WriteHeader error received: RC = 0x%08X Expected = %d File = '%s''

##### Global [MM\_CFS\_COMPUTECRCFROMFILE\_ERR\_EID](#AAAAAAAAFR)

'CFS\_ComputeCRCFromFile error received: RC = 0x%08X File = '%s''

##### Global [MM\_CMD\_FNAME\_ERR\_EID](#AAAAAAAAFS)

'Command specified filename invalid: Name = '%s''

##### Global [MM\_CMD\_SUB\_ERR\_EID](#AAAAAAAAGN)

'Error Subscribing to MM Command, RC = 0x%08X'

##### Global [MM\_CR\_PIPE\_ERR\_EID](#AAAAAAAAGL)

'Error Creating SB Pipe, RC = 0x%08X'

##### Global [MM\_DATA\_SIZE\_BITS\_ERR\_EID](#AAAAAAAAFB)

'Data size in bits invalid: Data Size = %d'

##### Global [MM\_DATA\_SIZE\_BYTES\_ERR\_EID](#AAAAAAAAFA)

'Data size in bytes invalid or exceeds limits: Data Size = %d'

##### Global [MM\_DMP\_MEM\_FILE\_INF\_EID](#AAAAAAAAES)

'Dump Memory To File Command: Dumped %d bytes from address 0x%08X to file '%s''

##### Global [MM\_DUMP\_INEVENT\_INF\_EID](#AAAAAAAAET)

'Memory Dump: 0x%02X from address: 0x%08lX'

##### Global [MM\_EEPROM\_WRITE\_DIS\_ERR\_EID](#AAAAAAAAGH)

'Error requesting EEPROM bank d write disable, cFE\_Status= 0xX'

##### Global [MM\_EEPROM\_WRITE\_DIS\_INF\_EID](#AAAAAAAAGG)

'EEPROM bank d write disabled, cFE\_Status= 0xX'

##### Global [MM\_EEPROM\_WRITE\_ENA\_ERR\_EID](#AAAAAAAAGF)

'Error requesting EEPROM bank d write enable, cFE\_Status= 0xX'

##### Global [MM\_EEPROM\_WRITE\_ENA\_INF\_EID](#AAAAAAAAGE)

'EEPROM bank d write enabled, cFE\_Status= 0xX'

##### Global [MM\_FILE\_LOAD\_PARAMS\_ERR\_EID](#AAAAAAAAFU)

'Load file failed parameters check: File = '%s''

##### Global [MM\_FILL\_INF\_EID](#AAAAAAAAEL)

'Fill Memory Command: Filled %d bytes at address: 0x%08X with pattern: 0x%08X'

##### Global [MM\_FILL\_MEM16\_ALIGN\_WARN\_INF\_EID](#AAAAAAAAGP)

'MM\_FillMem16 NumOfBytes not multiple of 2. Reducing from d to d.'

##### Global [MM\_FILL\_MEM32\_ALIGN\_WARN\_INF\_EID](#AAAAAAAAGO)

'MM\_FillMem32 NumOfBytes not multiple of 4. Reducing from d to d.'

##### Global [MM\_HK\_SUB\_ERR\_EID](#AAAAAAAAGM)

'Error Subscribing to HK Request, RC = 0x%08X'

##### Global [MM\_HKREQ\_LEN\_ERR\_EID](#AAAAAAAAFX)

'Invalid HK request msg length: ID = 0x%04X, CC = %d, Len = %d, Expected = %d'

##### Global [MM\_INIT\_INF\_EID](#AAAAAAAAEG)

'MM Initialized. Version %d.%d.%d.%d'

##### Global [MM\_LD\_FILE\_SIZE\_ERR\_EID](#AAAAAAAAFT)

'Load file size error: Reported by OS = %d Expected = %d File = '%s''

##### Global [MM\_LD\_MEM\_FILE\_INF\_EID](#AAAAAAAAEK)

'Load Memory From File Command: Loaded %d bytes to address 0x%08X from file '%s''

##### Global [MM\_LEN\_ERR\_EID](#AAAAAAAAEX)

'Invalid msg length: ID = 0x%04X, CC = %d, Len = %d, Expected = %d'

##### Global [MM\_LOAD\_FILE\_CRC\_ERR\_EID](#AAAAAAAAFF)

'Load file CRC failure: Expected = 0x%X Calculated = 0x%X File = '%s''

##### Global [MM\_LOAD\_WID\_CRC\_ERR\_EID](#AAAAAAAAFG)

'Interrupts Disabled Load CRC failure: Expected = 0x%X Calculated = 0x%X'

##### Global [MM\_LOAD\_WID\_INF\_EID](#AAAAAAAAEJ)

'Load Memory WID Command: Wrote %d bytes to address: 0x%08X'

##### Global [MM\_MEMTYPE\_ERR\_EID](#AAAAAAAAEY)

'Invalid memory type specified: MemType = %d'

##### Global [MM\_MID\_ERR\_EID](#AAAAAAAAEV)

'Invalid command pipe message ID: 0x%X'

##### Global [MM\_NOOP\_INF\_EID](#AAAAAAAAEH)

'No-op command. Version %d.%d.%d.%d'

##### Global [MM\_OS\_CLOSE\_ERR\_EID](#AAAAAAAAFM)

'OS\_close error received: RC = 0x%08X File = '%s''

##### Global [MM\_OS\_CREAT\_ERR\_EID](#AAAAAAAAFK)

'OS\_OpenCreate error received: RC = 0x%08X File = '%s''

##### Global [MM\_OS\_EEPROMWRITE16\_ERR\_EID](#AAAAAAAAFI)

'OS\_EepromWrite16 error received: RC = 0x%08X Addr = 0x%08X'

##### Global [MM\_OS\_EEPROMWRITE32\_ERR\_EID](#AAAAAAAAFJ)

'OS\_EepromWrite32 error received: RC = 0x%08X Addr = 0x%08X'

##### Global [MM\_OS\_EEPROMWRITE8\_ERR\_EID](#AAAAAAAAFH)

'OS\_EepromWrite8 error received: RC = 0x%08X Addr = 0x%08X'

##### Global [MM\_OS\_MEMVALIDATE\_ERR\_EID](#AAAAAAAAFE)

'CFE\_PSP\_MemValidate error received: RC = 0x%08X Addr = 0x%08X Size = %d MemType = %d'

##### Global [MM\_OS\_OPEN\_ERR\_EID](#AAAAAAAAFL)

'OS\_OpenCreate error received: RC = 0x%08X File = '%s''

##### Global [MM\_OS\_READ\_ERR\_EID](#AAAAAAAAFN)

'OS\_read error received: RC = 0x%08X File = '%s''

##### Global [MM\_OS\_READ\_EXP\_ERR\_EID](#AAAAAAAAFO)

'OS\_read error received: RC = 0x%08X Expected = %d File = '%s''

##### Global [MM\_OS\_STAT\_ERR\_EID](#AAAAAAAAFQ)

'OS\_stat error received: RC = 0x%08X File = '%s''

##### Global [MM\_OS\_WRITE\_EXP\_ERR\_EID](#AAAAAAAAFP)

'OS\_write error received: RC = 0x%08X Expected = %d File = '%s''

##### Global [MM\_OS\_ZERO\_READ\_ERR\_EID](#AAAAAAAAGI)

'Zero bytes read by OS\_read of file '%s''

##### Global [MM\_PEEK\_BYTE\_INF\_EID](#AAAAAAAAEM)

'Peek Command: Addr = 0x%08X Size = 8 bits Data = 0x%02X'

##### Global [MM\_PEEK\_DWORD\_INF\_EID](#AAAAAAAAEO)

'Peek Command: Addr = 0x%08X Size = 32 bits Data = 0x%08X'

##### Global [MM\_PEEK\_WORD\_INF\_EID](#AAAAAAAAEN)

'Peek Command: Addr = 0x%08X Size = 16 bits Data = 0x%04X'

##### Global [MM\_PIPE\_ERR\_EID](#AAAAAAAAEU)

'SB Pipe Read Error, App will exit. RC = 0x%08X'

##### Global [MM\_POKE\_BYTE\_INF\_EID](#AAAAAAAAEP)

'Poke Command: Addr = 0x%08X Size = 8 bits Data = 0x%02X'

##### Global [MM\_POKE\_DWORD\_INF\_EID](#AAAAAAAAER)

'Poke Command: Addr = 0x%08X Size = 32 bits Data = 0x%08X'

##### Global [MM\_POKE\_WORD\_INF\_EID](#AAAAAAAAEQ)

'Poke Command: Addr = 0x%08X Size = 16 bits Data = 0x%04X'

##### Global [MM\_PSP\_READ\_ERR\_EID](#AAAAAAAAGJ)

'PSP read memory error: RC=0x%08X, Src=0x%08X, Tgt=0x%08X, Type='s''

##### Global [MM\_PSP\_WRITE\_ERR\_EID](#AAAAAAAAGK)

'PSP write memory error: RC=0x%08X, Address=0x%08X, MemType='s''

##### Global [MM\_RESET\_INF\_EID](#AAAAAAAAEI)

'Reset counters command received'

##### Global [MM\_SYM\_LOOKUP\_INF\_EID](#AAAAAAAAFY)

'Symbol Lookup Command: Name = '%s' Addr = 0x%08X'

##### Global [MM\_SYMFILENAME\_NUL\_ERR\_EID](#AAAAAAAAGB)

'NUL (empty) string specified as symbol dump file name'

##### Global [MM\_SYMNAME\_ERR\_EID](#AAAAAAAAEZ)

'Symbolic address can't be resolved: Name = '%s''

##### Global [MM\_SYMNAME\_NUL\_ERR\_EID](#AAAAAAAAFZ)

'NUL (empty) string specified as symbol name'

##### Global [MM\_SYMTBL\_TO\_FILE\_FAIL\_ERR\_EID](#AAAAAAAAGC)

'Error dumping symbol table, OS Status= 0xX, File='s''

##### Global [MM\_SYMTBL\_TO\_FILE\_INF\_EID](#AAAAAAAAGA)

'Symbol Table Dump to File Started: Name = '%s''

##### Global [MM\_SYMTBL\_TO\_FILE\_INVALID\_ERR\_EID](#AAAAAAAAGD)

'Illegal characters in target filename, File='s''

# Data Structure Documentation

## MM\_AppData\_t Struct Reference

#include <mm\_app.h>

### Data Fields

* [MM\_HkPacket\_t](#AAAAAAAAIY) [HkPacket](#AAAAAAAAPU)

*Housekeeping telemetry packet.*

* CFE\_SB\_PipeId\_t [CmdPipe](#AAAAAAAAPV)

*Command pipe ID.*

* uint32 [RunStatus](#AAAAAAAAPW)

*Application run status.*

* uint32 [LoadBuffer](#AAAAAAAAPX) [[MM\_MAX\_LOAD\_DATA\_SEG](#AAAAAAAAAW)/4]

*Load file i/o buffer.*

* uint32 [DumpBuffer](#AAAAAAAAPY) [[MM\_MAX\_DUMP\_DATA\_SEG](#AAAAAAAABA)/4]

*Dump file i/o buffer.*

* uint32 [FillBuffer](#AAAAAAAAPZ) [[MM\_MAX\_FILL\_DATA\_SEG](#AAAAAAAABD)/4]

*Fill memory buffer.*

### Detailed Description

MM global data structure.

### Field Documentation

#### CFE\_SB\_PipeId\_t MM\_AppData\_t::CmdPipe

Command pipe ID.

#### uint32 MM\_AppData\_t::DumpBuffer[[MM\_MAX\_DUMP\_DATA\_SEG](#AAAAAAAABA)/4]

Dump file i/o buffer.

#### uint32 MM\_AppData\_t::FillBuffer[[MM\_MAX\_FILL\_DATA\_SEG](#AAAAAAAABD)/4]

Fill memory buffer.

#### [MM\_HkPacket\_t](#AAAAAAAAIY) MM\_AppData\_t::HkPacket

Housekeeping telemetry packet.

#### uint32 MM\_AppData\_t::LoadBuffer[[MM\_MAX\_LOAD\_DATA\_SEG](#AAAAAAAAAW)/4]

Load file i/o buffer.

#### uint32 MM\_AppData\_t::RunStatus

Application run status.

#### The documentation for this struct was generated from the following file:

fsw/src/[mm\_app.h](#AAAAAAAACL)

## MM\_DumpInEventCmd\_t Struct Reference

#include <mm\_msg.h>

### Data Fields

* CFE\_MSG\_CommandHeader\_t [CmdHeader](#AAAAAAAAQA)
* uint8 [MemType](#AAAAAAAAQB)

*Memory dump type.*

* uint8 [NumOfBytes](#AAAAAAAAQC)

*Number of bytes to be dumped.*

* uint16 [Padding](#AAAAAAAAQD)

*Structure padding.*

* CFS\_SymAddr\_t [SrcSymAddress](#AAAAAAAAQE)

*Symbolic source address.*

### Detailed Description

Dump Memory In Event Message Command For command details see [MM\_DUMP\_IN\_EVENT\_CC](#AAAAAAAADT).

### Field Documentation

#### CFE\_MSG\_CommandHeader\_t MM\_DumpInEventCmd\_t::CmdHeader

#### uint8 MM\_DumpInEventCmd\_t::MemType

Memory dump type.

#### uint8 MM\_DumpInEventCmd\_t::NumOfBytes

Number of bytes to be dumped.

#### uint16 MM\_DumpInEventCmd\_t::Padding

Structure padding.

#### CFS\_SymAddr\_t MM\_DumpInEventCmd\_t::SrcSymAddress

Symbolic source address.

#### The documentation for this struct was generated from the following file:

fsw/src/[mm\_msg.h](#AAAAAAAAIQ)

## MM\_DumpMemToFileCmd\_t Struct Reference

#include <mm\_msg.h>

### Data Fields

* CFE\_MSG\_CommandHeader\_t [CmdHeader](#AAAAAAAAQF)
* uint8 [MemType](#AAAAAAAAQG)

*Memory dump type.*

* uint8 [Padding](#AAAAAAAAQH) [3]

*Structure padding.*

* uint32 [NumOfBytes](#AAAAAAAAQI)

*Number of bytes to be dumped.*

* CFS\_SymAddr\_t [SrcSymAddress](#AAAAAAAAQJ)

*Symbol plus optional offset.*

* char [FileName](#AAAAAAAAQK) [OS\_MAX\_PATH\_LEN]

*Name of memory dump file.*

### Detailed Description

Memory Dump To File Command For command details see [MM\_DUMP\_MEM\_TO\_FILE\_CC](#AAAAAAAADV).

### Field Documentation

#### CFE\_MSG\_CommandHeader\_t MM\_DumpMemToFileCmd\_t::CmdHeader

#### char MM\_DumpMemToFileCmd\_t::FileName[OS\_MAX\_PATH\_LEN]

Name of memory dump file.

#### uint8 MM\_DumpMemToFileCmd\_t::MemType

Memory dump type.

#### uint32 MM\_DumpMemToFileCmd\_t::NumOfBytes

Number of bytes to be dumped.

#### uint8 MM\_DumpMemToFileCmd\_t::Padding[3]

Structure padding.

#### CFS\_SymAddr\_t MM\_DumpMemToFileCmd\_t::SrcSymAddress

Symbol plus optional offset.

#### The documentation for this struct was generated from the following file:

fsw/src/[mm\_msg.h](#AAAAAAAAIQ)

## MM\_EepromWriteDisCmd\_t Struct Reference

#include <mm\_msg.h>

### Data Fields

* CFE\_MSG\_CommandHeader\_t [CmdHeader](#AAAAAAAAQL)
* uint32 [Bank](#AAAAAAAAQM)

*EEPROM bank number to write-disable.*

### Detailed Description

EEPROM Write Disable Command For command details see [MM\_DISABLE\_EEPROM\_WRITE\_CC](#AAAAAAAACF).

### Field Documentation

#### uint32 MM\_EepromWriteDisCmd\_t::Bank

EEPROM bank number to write-disable.

#### CFE\_MSG\_CommandHeader\_t MM\_EepromWriteDisCmd\_t::CmdHeader

#### The documentation for this struct was generated from the following file:

fsw/src/[mm\_msg.h](#AAAAAAAAIQ)

## MM\_EepromWriteEnaCmd\_t Struct Reference

#include <mm\_msg.h>

### Data Fields

* CFE\_MSG\_CommandHeader\_t [CmdHeader](#AAAAAAAAQN)
* uint32 [Bank](#AAAAAAAAQO)

*EEPROM bank number to write-enable.*

### Detailed Description

EEPROM Write Enable Command For command details see [MM\_ENABLE\_EEPROM\_WRITE\_CC](#AAAAAAAACG).

### Field Documentation

#### uint32 MM\_EepromWriteEnaCmd\_t::Bank

EEPROM bank number to write-enable.

#### CFE\_MSG\_CommandHeader\_t MM\_EepromWriteEnaCmd\_t::CmdHeader

#### The documentation for this struct was generated from the following file:

fsw/src/[mm\_msg.h](#AAAAAAAAIQ)

## MM\_FillMemCmd\_t Struct Reference

#include <mm\_msg.h>

### Data Fields

* CFE\_MSG\_CommandHeader\_t [CmdHeader](#AAAAAAAAQP)
* uint8 [MemType](#AAAAAAAAQQ)

*Memory type.*

* uint8 [Padding](#AAAAAAAAQR) [3]

*Structure padding.*

* uint32 [NumOfBytes](#AAAAAAAAQS)

*Number of bytes to fill.*

* uint32 [FillPattern](#AAAAAAAAQT)

*Fill pattern to use.*

* CFS\_SymAddr\_t [DestSymAddress](#AAAAAAAAQU)

*Symbol plus optional offset.*

### Detailed Description

Memory Fill Command For command details see [MM\_FILL\_MEM\_CC](#AAAAAAAAHL).

### Field Documentation

#### CFE\_MSG\_CommandHeader\_t MM\_FillMemCmd\_t::CmdHeader

#### CFS\_SymAddr\_t MM\_FillMemCmd\_t::DestSymAddress

Symbol plus optional offset.

#### uint32 MM\_FillMemCmd\_t::FillPattern

Fill pattern to use.

#### uint8 MM\_FillMemCmd\_t::MemType

Memory type.

#### uint32 MM\_FillMemCmd\_t::NumOfBytes

Number of bytes to fill.

#### uint8 MM\_FillMemCmd\_t::Padding[3]

Structure padding.

#### The documentation for this struct was generated from the following file:

fsw/src/[mm\_msg.h](#AAAAAAAAIQ)

## MM\_HkPacket\_t Struct Reference

#include <mm\_msg.h>

### Data Fields

* CFE\_MSG\_TelemetryHeader\_t [TlmHeader](#AAAAAAAAQV)

*cFE SB Tlm Msg Hdr*

* uint8 [CmdCounter](#AAAAAAAAMZ)

*MM Application Command Counter.*

* uint8 [ErrCounter](#AAAAAAAAND)

*MM Application Command Error Counter.*

* uint8 [LastAction](#AAAAAAAANH)

*Last command action executed.*

* uint8 [MemType](#AAAAAAAANJ)

*Memory type for last command.*

* cpuaddr [Address](#AAAAAAAAMV)

*Fully resolved address used for last command.*

* uint32 [DataValue](#AAAAAAAANB)

*Last command data value – may be fill pattern or peek/poke value.*

* uint32 [BytesProcessed](#AAAAAAAAMX)

*Bytes processed for last command.*

* char [FileName](#AAAAAAAANF) [OS\_MAX\_PATH\_LEN]

*Name of the data file used for last command, where applicable.*

### Detailed Description

##### Purpose:

Housekeeping Packet Structure

### Field Documentation

#### cpuaddr MM\_HkPacket\_t::Address

Fully resolved address used for last command.

#### uint32 MM\_HkPacket\_t::BytesProcessed

Bytes processed for last command.

#### uint8 MM\_HkPacket\_t::CmdCounter

MM Application Command Counter.

#### uint32 MM\_HkPacket\_t::DataValue

Last command data value – may be fill pattern or peek/poke value.

#### uint8 MM\_HkPacket\_t::ErrCounter

MM Application Command Error Counter.

#### char MM\_HkPacket\_t::FileName[OS\_MAX\_PATH\_LEN]

Name of the data file used for last command, where applicable.

#### uint8 MM\_HkPacket\_t::LastAction

Last command action executed.

#### uint8 MM\_HkPacket\_t::MemType

Memory type for last command.

#### CFE\_MSG\_TelemetryHeader\_t MM\_HkPacket\_t::TlmHeader

cFE SB Tlm Msg Hdr

#### The documentation for this struct was generated from the following file:

fsw/src/[mm\_msg.h](#AAAAAAAAIQ)

## MM\_LoadDumpFileHeader\_t Struct Reference

#include <mm\_filedefs.h>

### Data Fields

* CFS\_SymAddr\_t [SymAddress](#AAAAAAAAQW)

*Symbolic load address or fully resolved dump address.*

* uint32 [NumOfBytes](#AAAAAAAAQX)

*Bytes to load or bytes dumped.*

* uint32 [Crc](#AAAAAAAAQY)

*CRC value for load or dump data.*

* uint8 [MemType](#AAAAAAAAQZ)

*Memory type used.*

* uint8 [Spare](#AAAAAAAARA) [3]

*Structure Padding.*

### Detailed Description

MM Load and Dump file header structure We use the same header structure for both dump and load files so a dump file can be reloaded back into memory if desired (providing a memory save and restore capability). This MM header is the secondary header, the standard cFE file header is primary header for all load and dump files.

### Field Documentation

#### uint32 MM\_LoadDumpFileHeader\_t::Crc

CRC value for load or dump data.

#### uint8 MM\_LoadDumpFileHeader\_t::MemType

Memory type used.

#### uint32 MM\_LoadDumpFileHeader\_t::NumOfBytes

Bytes to load or bytes dumped.

#### uint8 MM\_LoadDumpFileHeader\_t::Spare[3]

Structure Padding.

#### CFS\_SymAddr\_t MM\_LoadDumpFileHeader\_t::SymAddress

Symbolic load address or fully resolved dump address.

#### The documentation for this struct was generated from the following file:

fsw/src/[mm\_filedefs.h](#AAAAAAAAGV)

## MM\_LoadMemFromFileCmd\_t Struct Reference

#include <mm\_msg.h>

### Data Fields

* CFE\_MSG\_CommandHeader\_t [CmdHeader](#AAAAAAAARB)
* char [FileName](#AAAAAAAARC) [OS\_MAX\_PATH\_LEN]

*Name of memory load file.*

### Detailed Description

Memory Load From File Command For command details see [MM\_LOAD\_MEM\_FROM\_FILE\_CC](#AAAAAAAAHN).

### Field Documentation

#### CFE\_MSG\_CommandHeader\_t MM\_LoadMemFromFileCmd\_t::CmdHeader

#### char MM\_LoadMemFromFileCmd\_t::FileName[OS\_MAX\_PATH\_LEN]

Name of memory load file.

#### The documentation for this struct was generated from the following file:

fsw/src/[mm\_msg.h](#AAAAAAAAIQ)

## MM\_LoadMemWIDCmd\_t Struct Reference

#include <mm\_msg.h>

### Data Fields

* CFE\_MSG\_CommandHeader\_t [CmdHeader](#AAAAAAAARD)
* uint8 [NumOfBytes](#AAAAAAAARE)

*Number of bytes to be loaded.*

* uint8 [Padding](#AAAAAAAARF) [3]

*Structure padding.*

* uint32 [Crc](#AAAAAAAARG)

*Data check value.*

* CFS\_SymAddr\_t [DestSymAddress](#AAAAAAAARH)

*Symbolic destination load address.*

* uint8 [DataArray](#AAAAAAAARI) [[MM\_MAX\_UNINTERRUPTIBLE\_DATA](#AAAAAAAAAV)]

*Data to be loaded.*

### Detailed Description

Memory Load With Interrupts Disabled Command For command details see [MM\_LOAD\_MEM\_WID\_CC](#AAAAAAAAAF).

### Field Documentation

#### CFE\_MSG\_CommandHeader\_t MM\_LoadMemWIDCmd\_t::CmdHeader

#### uint32 MM\_LoadMemWIDCmd\_t::Crc

Data check value.

#### uint8 MM\_LoadMemWIDCmd\_t::DataArray[[MM\_MAX\_UNINTERRUPTIBLE\_DATA](#AAAAAAAAAV)]

Data to be loaded.

#### CFS\_SymAddr\_t MM\_LoadMemWIDCmd\_t::DestSymAddress

Symbolic destination load address.

#### uint8 MM\_LoadMemWIDCmd\_t::NumOfBytes

Number of bytes to be loaded.

#### uint8 MM\_LoadMemWIDCmd\_t::Padding[3]

Structure padding.

#### The documentation for this struct was generated from the following file:

fsw/src/[mm\_msg.h](#AAAAAAAAIQ)

## MM\_LookupSymCmd\_t Struct Reference

#include <mm\_msg.h>

### Data Fields

* CFE\_MSG\_CommandHeader\_t [CmdHeader](#AAAAAAAARJ)
* char [SymName](#AAAAAAAARK) [OS\_MAX\_SYM\_LEN]

*Symbol name string.*

### Detailed Description

Symbol Table Lookup Command For command details see [MM\_LOOKUP\_SYM\_CC](#AAAAAAAACH).

### Field Documentation

#### CFE\_MSG\_CommandHeader\_t MM\_LookupSymCmd\_t::CmdHeader

#### char MM\_LookupSymCmd\_t::SymName[OS\_MAX\_SYM\_LEN]

Symbol name string.

#### The documentation for this struct was generated from the following file:

fsw/src/[mm\_msg.h](#AAAAAAAAIQ)

## MM\_NoArgsCmd\_t Struct Reference

#include <mm\_msg.h>

### Data Fields

* CFE\_MSG\_CommandHeader\_t [CmdHeader](#AAAAAAAARL)

### Detailed Description

No Arguments Command For command details see [MM\_NOOP\_CC](#AAAAAAAACI), [MM\_RESET\_CC](#AAAAAAAACJ).

### Field Documentation

#### CFE\_MSG\_CommandHeader\_t MM\_NoArgsCmd\_t::CmdHeader

#### The documentation for this struct was generated from the following file:

fsw/src/[mm\_msg.h](#AAAAAAAAIQ)

## MM\_PeekCmd\_t Struct Reference

#include <mm\_msg.h>

### Data Fields

* CFE\_MSG\_CommandHeader\_t [CmdHeader](#AAAAAAAARM)
* uint8 [DataSize](#AAAAAAAARN)

*Size of the data to be read.*

* uint8 [MemType](#AAAAAAAARO)

*Memory type to peek data from.*

* uint8 [Padding](#AAAAAAAARP) [2]

*Structure padding.*

* CFS\_SymAddr\_t [SrcSymAddress](#AAAAAAAARQ)

*Symbolic source peek address.*

### Detailed Description

Memory Peek Command For command details see [MM\_PEEK\_CC](#AAAAAAAADX).

### Field Documentation

#### CFE\_MSG\_CommandHeader\_t MM\_PeekCmd\_t::CmdHeader

#### uint8 MM\_PeekCmd\_t::DataSize

Size of the data to be read.

#### uint8 MM\_PeekCmd\_t::MemType

Memory type to peek data from.

#### uint8 MM\_PeekCmd\_t::Padding[2]

Structure padding.

#### CFS\_SymAddr\_t MM\_PeekCmd\_t::SrcSymAddress

Symbolic source peek address.

#### The documentation for this struct was generated from the following file:

fsw/src/[mm\_msg.h](#AAAAAAAAIQ)

## MM\_PokeCmd\_t Struct Reference

#include <mm\_msg.h>

### Data Fields

* CFE\_MSG\_CommandHeader\_t [CmdHeader](#AAAAAAAARR)
* uint8 [DataSize](#AAAAAAAARS)

*Size of the data to be written.*

* uint8 [MemType](#AAAAAAAART)

*Memory type to poke data to.*

* uint8 [Padding](#AAAAAAAARU) [2]

*Structure padding.*

* uint32 [Data](#AAAAAAAARV)

*Data to be written.*

* CFS\_SymAddr\_t [DestSymAddress](#AAAAAAAARW)

*Symbolic destination poke address.*

### Detailed Description

Memory Poke Command For command details see [MM\_POKE\_CC](#AAAAAAAAHO).

### Field Documentation

#### CFE\_MSG\_CommandHeader\_t MM\_PokeCmd\_t::CmdHeader

#### uint32 MM\_PokeCmd\_t::Data

Data to be written.

#### uint8 MM\_PokeCmd\_t::DataSize

Size of the data to be written.

#### CFS\_SymAddr\_t MM\_PokeCmd\_t::DestSymAddress

Symbolic destination poke address.

#### uint8 MM\_PokeCmd\_t::MemType

Memory type to poke data to.

#### uint8 MM\_PokeCmd\_t::Padding[2]

Structure padding.

#### The documentation for this struct was generated from the following file:

fsw/src/[mm\_msg.h](#AAAAAAAAIQ)

## MM\_SymTblToFileCmd\_t Struct Reference

#include <mm\_msg.h>

### Data Fields

* CFE\_MSG\_CommandHeader\_t [CmdHeader](#AAAAAAAARX)
* char [FileName](#AAAAAAAARY) [OS\_MAX\_PATH\_LEN]

*Name of symbol dump file.*

### Detailed Description

Save Symbol Table To File Command For command details see [MM\_SYMTBL\_TO\_FILE\_CC](#AAAAAAAACK).

### Field Documentation

#### CFE\_MSG\_CommandHeader\_t MM\_SymTblToFileCmd\_t::CmdHeader

#### char MM\_SymTblToFileCmd\_t::FileName[OS\_MAX\_PATH\_LEN]

Name of symbol dump file.

#### The documentation for this struct was generated from the following file:

fsw/src/[mm\_msg.h](#AAAAAAAAIQ)

# File Documentation

## fsw/mission\_inc/mm\_mission\_cfg.h File Reference

#include "cfe\_mission\_cfg.h"

### Macros

* #define [MM\_LOAD\_WID\_CRC\_TYPE](#AAAAAAAAAC)  CFE\_MISSION\_ES\_DEFAULT\_CRC
* #define [MM\_LOAD\_FILE\_CRC\_TYPE](#AAAAAAAAAD)  CFE\_MISSION\_ES\_DEFAULT\_CRC
* #define [MM\_DUMP\_FILE\_CRC\_TYPE](#AAAAAAAAAE)  CFE\_MISSION\_ES\_DEFAULT\_CRC

### Macro Definition Documentation

#### #define MM\_DUMP\_FILE\_CRC\_TYPE  CFE\_MISSION\_ES\_DEFAULT\_CRC

##### Purpose:

CRC type for dump files

##### Description:

CFE CRC type to use when processing memory dumps to a file.

##### Limits:

This must be one of the CRC types supported by the #CFE\_ES\_CalculateCRC function.

#### #define MM\_LOAD\_FILE\_CRC\_TYPE  CFE\_MISSION\_ES\_DEFAULT\_CRC

##### Purpose:

CRC type for load files

##### Description:

CFE CRC type to use when processing memory loads from a file.

##### Limits:

This must be one of the CRC types supported by the #CFE\_ES\_CalculateCRC function.

#### #define MM\_LOAD\_WID\_CRC\_TYPE  CFE\_MISSION\_ES\_DEFAULT\_CRC

##### Purpose:

CRC type for interrupts disabled loads

##### Description:

CFE CRC type to use when processing the "memory load with interrupts disabled" ([MM\_LOAD\_MEM\_WID\_CC](#AAAAAAAAAF)) command.

##### Limits:

This must be one of the CRC types supported by the #CFE\_ES\_CalculateCRC function.

## fsw/mission\_inc/mm\_perfids.h File Reference

### Macros

MM CFS Application Performance IDs

#define [MM\_APPMAIN\_PERF\_ID](#AAAAAAAAAH)  30

#define [MM\_SEGBREAK\_PERF\_ID](#AAAAAAAAAI)  31

#define [MM\_EEPROM\_POKE\_PERF\_ID](#AAAAAAAAAJ)  32

#define [MM\_EEPROM\_FILELOAD\_PERF\_ID](#AAAAAAAAAK)  33

#define [MM\_EEPROM\_FILL\_PERF\_ID](#AAAAAAAAAL)  34

### Macro Definition Documentation

#### #define MM\_APPMAIN\_PERF\_ID  30

#### #define MM\_EEPROM\_FILELOAD\_PERF\_ID  33

#### #define MM\_EEPROM\_FILL\_PERF\_ID  34

#### #define MM\_EEPROM\_POKE\_PERF\_ID  32

#### #define MM\_SEGBREAK\_PERF\_ID  31

#### 

## fsw/platform\_inc/mm\_msgids.h File Reference

### Macros

MM Command Message IDs

s#define [MM\_CMD\_MID](#AAAAAAAAAN)  0x1888

*Msg ID for cmds to mm.*

#define [MM\_SEND\_HK\_MID](#AAAAAAAAAO)  0x1889

*Msg ID to request mm HK.*

MM Telemetry Message ID

#define [MM\_HK\_TLM\_MID](#AAAAAAAAAP)  0x0887

*MM Housekeeping Telemetry.*

### Macro Definition Documentation

#### define MM\_CMD\_MID  0x1888

Msg ID for cmds to mm.

#### #define MM\_HK\_TLM\_MID  0x0887

MM Housekeeping Telemetry.

#### #define MM\_SEND\_HK\_MID  0x1889

Msg ID to request mm HK.

## fsw/platform\_inc/mm\_platform\_cfg.h File Reference

### Macros

* #define [MM\_CFE\_HDR\_SUBTYPE](#AAAAAAAAAR)  0x4D4D5354
* #define [MM\_CFE\_HDR\_DESCRIPTION](#AAAAAAAAAS)  "Memory Manager dump file"
* #define [MM\_MAX\_LOAD\_FILE\_DATA\_RAM](#AAAAAAAAAT)  (1024\*1024)
* #define [MM\_MAX\_LOAD\_FILE\_DATA\_EEPROM](#AAAAAAAAAU)  (128\*1024)
* #define [MM\_MAX\_UNINTERRUPTIBLE\_DATA](#AAAAAAAAAV)  200
* #define [MM\_MAX\_LOAD\_DATA\_SEG](#AAAAAAAAAW)  200
* #define [MM\_MAX\_DUMP\_FILE\_DATA\_RAM](#AAAAAAAAAX)  (1024\*1024)
* #define [MM\_MAX\_DUMP\_FILE\_DATA\_EEPROM](#AAAAAAAAAY)  (128\*1024)
* #define [MM\_MAX\_DUMP\_FILE\_DATA\_SYMTBL](#AAAAAAAAAZ)  (128\*1024)
* #define [MM\_MAX\_DUMP\_DATA\_SEG](#AAAAAAAABA)  200
* #define [MM\_MAX\_FILL\_DATA\_RAM](#AAAAAAAABB)  (1024\*1024)
* #define [MM\_MAX\_FILL\_DATA\_EEPROM](#AAAAAAAABC)  (128\*1024)
* #define [MM\_MAX\_FILL\_DATA\_SEG](#AAAAAAAABD)  200
* #define [MM\_OPT\_CODE\_MEM32\_MEMTYPE](#AAAAAAAABE)
* #define [MM\_MAX\_LOAD\_FILE\_DATA\_MEM32](#AAAAAAAABF)  (1024\*1024)
* #define [MM\_MAX\_DUMP\_FILE\_DATA\_MEM32](#AAAAAAAABG)  (1024\*1024)
* #define [MM\_MAX\_FILL\_DATA\_MEM32](#AAAAAAAABH)  (1024\*1024)
* #define [MM\_OPT\_CODE\_MEM16\_MEMTYPE](#AAAAAAAABI)
* #define [MM\_MAX\_LOAD\_FILE\_DATA\_MEM16](#AAAAAAAABJ)  (1024\*1024)
* #define [MM\_MAX\_DUMP\_FILE\_DATA\_MEM16](#AAAAAAAABK)  (1024\*1024)
* #define [MM\_MAX\_FILL\_DATA\_MEM16](#AAAAAAAABL)  (1024\*1024)
* #define [MM\_OPT\_CODE\_MEM8\_MEMTYPE](#AAAAAAAABM)
* #define [MM\_MAX\_LOAD\_FILE\_DATA\_MEM8](#AAAAAAAABN)  (1024\*1024)
* #define [MM\_MAX\_DUMP\_FILE\_DATA\_MEM8](#AAAAAAAABO)  (1024\*1024)
* #define [MM\_MAX\_FILL\_DATA\_MEM8](#AAAAAAAABP)  (1024\*1024)
* #define [MM\_PROCESSOR\_CYCLE](#AAAAAAAABQ)  0
* #define [MM\_MISSION\_REV](#AAAAAAAABR)  0

### Macro Definition Documentation

#### #define MM\_CFE\_HDR\_DESCRIPTION  "Memory Manager dump file"

##### Purpose:

Memory Management File – cFE file header description

##### Description:

This parameter defines the text string that may be used to identify Memory Management files.

##### Limits:

The string length (including string terminator) cannot exceed #CFE\_FS\_HDR\_DESC\_MAX\_LEN. (limit is not verified)

#### #define MM\_CFE\_HDR\_SUBTYPE  0x4D4D5354

##### Purpose:

Memory Managment File – cFE file header sub-type

##### Description:

This parameter defines the value that is used to identify a Memory Management file.

##### Limits:

The file header data type for the value is 32 bits unsigned, thus the value can be anything from zero to 4,294,967,295.

#### #define MM\_MAX\_DUMP\_DATA\_SEG  200

##### Purpose:

Maximum number of bytes per dump data segment

##### Description:

Maximum number of bytes MM will dump per task cycle to prevent CPU hogging (segmented dump).

##### Limits:

This parameter is limited only by the maximum value of the uint32 type. However, setting this value to a large number will decrease the amount of time available for other tasks to execute and increase MM CPU utilization during dump operations.

#### #define MM\_MAX\_DUMP\_FILE\_DATA\_EEPROM  (128\*1024)

##### Purpose:

Maximum number of bytes for a file dump from EEPROM memory

##### Description:

Maximum number of bytes that can be dumped from EEPROM into a single dump file.

##### Limits:

This parameter is limited only by the maximum value of the uint32 type. However, setting this value to a large number will increase the likelihood of MM being late responding to housekeeping requests since it cannot process such a request while a load or dump is in progress.

#### #define MM\_MAX\_DUMP\_FILE\_DATA\_MEM16  (1024\*1024)

##### Purpose:

Maximum number of bytes for a file dump from MEM16 memory

##### Description:

Maximum number of bytes that can be dumped from the optional MEM16 memory type to a single dump file.

##### Limits:

This value should be word aligned. Setting this value to a large number will increase the likelyhood of MM being late responding to housekeeping requests since it cannot process such a request while a load or dump is in progress.

#### #define MM\_MAX\_DUMP\_FILE\_DATA\_MEM32  (1024\*1024)

##### Purpose:

Maximum number of bytes for a file dump from MEM32 memory

##### Description:

Maximum number of bytes that can be dumped from the optional MEM32 memory type to a single dump file.

##### Limits:

This value should be longword aligned. This parameter is limited by the maximum value of the uint32 type. Setting this value to a large number will increase the likelihood of MM being late responding to housekeeping requests since it cannot process such a request while a load or dump is in progress.

#### #define MM\_MAX\_DUMP\_FILE\_DATA\_MEM8  (1024\*1024)

##### Purpose:

Maximum number of bytes for a file dump from MEM8 memory

##### Description:

Maximum number of bytes that can be dumped from the optional MEM8 memory type to a single dump file.

##### Limits:

This parameter is limited by the maximum value of the uint8 type. Setting this value to a large number will increase the likelihood of MM being late responding to housekeeping requests since it cannot process such a request while a load or dump is in progress.

#### #define MM\_MAX\_DUMP\_FILE\_DATA\_RAM  (1024\*1024)

##### Purpose:

Maximum number of bytes for a file dump from RAM memory

##### Description:

Maximum number of bytes that can be dumped from RAM into a single dump file.

##### Limits:

This parameter is limited only by the maximum value of the uint32 type. However, setting this value to a large number will increase the likelihood of MM being late responding to housekeeping requests since it cannot process such a request while a load or dump is in progress.

#### #define MM\_MAX\_DUMP\_FILE\_DATA\_SYMTBL  (128\*1024)

##### Purpose:

Maximum number of bytes for a symbol table file dump

##### Description:

Maximum number of bytes that can be dumped from the symbol table into a single dump file.

##### Limits:

This parameter is limited only by the maximum value of the uint32 type. However, setting this value to a large number will impact the OSAL since it is responsible for generating the dump file.

#### #define MM\_MAX\_FILL\_DATA\_EEPROM  (128\*1024)

##### Purpose:

Maximum number of bytes for a fill to EEPROM memory

##### Description:

Maximum number of bytes that can be loaded into EEPROM with a single memory fill command.

##### Limits:

This parameter is limited only by the maximum value of the uint32 type. However, setting this value to a large number will increase the likelihood of MM being late responding to housekeeping requests since it cannot process such a request while a fill operation is in progress.

#### #define MM\_MAX\_FILL\_DATA\_MEM16  (1024\*1024)

##### Purpose:

Maximum number of bytes for a fill to MEM16 memory

##### Description:

Maximum number of bytes that can be loaded into the optional MEM16 memory type with a single memory fill command.

##### Limits:

This value should be word aligned. This parameter is limited by the maximum value of the uint16 type. Setting this value to a large number will increase the likelihood of MM being late responding to housekeeping requests since it cannot process such a request while a memory fill operation is in progress.

#### #define MM\_MAX\_FILL\_DATA\_MEM32  (1024\*1024)

##### Purpose:

Maximum number of bytes for a fill to MEM32 memory

##### Description:

Maximum number of bytes that can be loaded into the optional MEM32 memory type with a single memory fill command.

##### Limits:

This value should be longword aligned. This parameter is limited by the maximum value of the uint16 type. Setting this value to a large number will increase the likelihood of MM being late responding to housekeeping requests since it cannot process such a request while a memory fill operation is in progress.

#### #define MM\_MAX\_FILL\_DATA\_MEM8  (1024\*1024)

##### Purpose:

Maximum number of bytes for a fill to MEM8 memory

##### Description:

Maximum number of bytes that can be loaded into the optional MEM8 memory type with a single memory fill command.

##### Limits:

This parameter is limited by the maximum value of the uint8 type. Setting this value to a large number will increase the likelihood of MM being late responding to housekeeping requests since it cannot process such a request while a memory fill operation is in progress.

#### #define MM\_MAX\_FILL\_DATA\_RAM  (1024\*1024)

##### Purpose:

Maximum number of bytes for a fill to RAM memory

##### Description:

Maximum number of bytes that can be loaded into RAM with a single memory fill command.

##### Limits:

This parameter is limited only by the maximum value of the uint32 type. However, setting this value to a large number will increase the likelihood of MM being late responding to housekeeping requests since it cannot process such a request while a fill operation is in progress.

#### #define MM\_MAX\_FILL\_DATA\_SEG  200

##### Purpose:

Maximum number of bytes per fill data segment

##### Description:

Maximum number of bytes MM will fill per task cycle to prevent CPU hogging (segmented fill).

##### Limits:

This parameter is limited only by the maximum value of the uint32 type. However, setting this value to a large number will decrease the amount of time available for other tasks to execute and increase MM CPU utilization during memory fill operations.

#### #define MM\_MAX\_LOAD\_DATA\_SEG  200

##### Purpose:

Maximum number of bytes per load data segment

##### Description:

Maximum number of bytes MM will load per task cycle to prevent CPU hogging (segmented load).

##### Limits:

The MM app does not place a limit on this parameter. However, setting this value to a large number will decrease the amount of time available for other tasks to execute and increase MM CPU utilization during load operations.

#### #define MM\_MAX\_LOAD\_FILE\_DATA\_EEPROM  (128\*1024)

##### Purpose:

Maximum number of bytes for a file load to EEPROM memory

##### Description:

Maximum number of bytes that can be loaded into EEPROM from a single load file.

##### Limits:

This parameter is limited only by the maximum value of the uint32 type. However, setting this value to a large number will increase the likelihood of MM being late responding to housekeeping requests since it cannot process such a request while a load or dump is in progress.

#### #define MM\_MAX\_LOAD\_FILE\_DATA\_MEM16  (1024\*1024)

##### Purpose:

Maximum number of bytes for a file load to MEM16 memory

##### Description:

Maximum number of bytes that can be loaded into the optional MEM16 memory type from a single load file.

##### Limits:

This value should be word aligned. This parameter is limited by the maximum value of the uint16 type. Setting this value to a large number will increase the likelihood of MM being late responding to housekeeping requests since it cannot process such a request while a load or dump is in progress.

#### #define MM\_MAX\_LOAD\_FILE\_DATA\_MEM32  (1024\*1024)

##### Purpose:

Maximum number of bytes for a file load to MEM32 memory

##### Description:

Maximum number of bytes that can be loaded into the optional MEM32 memory type from a single load file.

##### Limits:

This value should be longword aligned. This parameter is limited by the maximum value of the uint32 type. Setting this value to a large number will increase the likelihood of MM being late responding to housekeeping requests since it cannot process such a request while a load or dump is in progress.

#### #define MM\_MAX\_LOAD\_FILE\_DATA\_MEM8  (1024\*1024)

##### Purpose:

Maximum number of bytes for a file load to MEM8 memory

##### Description:

Maximum number of bytes that can be loaded into the optional MEM8 memory type from a single load file.

##### Limits:

This parameter is limited by the maximum value of the uint8 type. Setting this value to a large number will increase the likelihood of MM being late responding to housekeeping requests since it cannot process such a request while a load or dump is in progress.

#### #define MM\_MAX\_LOAD\_FILE\_DATA\_RAM  (1024\*1024)

##### Purpose:

Maximum number of bytes for a file load to RAM memory

##### Description:

Maximum number of bytes that can be loaded into RAM from a single load file.

##### Limits:

This parameter is limited only by the maximum value of the uint32 type. However, setting this value to a large number will increase the likelihood of MM being late responding to housekeeping requests since it cannot process such a request while a load or dump is in progress.

#### #define MM\_MAX\_UNINTERRUPTIBLE\_DATA  200

##### Purpose:

Maximum number of bytes for an uninterruptable load

##### Description:

Maximum number of bytes that can be loaded with the "memory load with interrupts disabled" ([MM\_LOAD\_MEM\_WID\_CC](#AAAAAAAAAF)) command.

##### Limits:

This parameter is limited to the size of a uint8 which is the data type used to specify the number of bytes to load in the command message.

If this data type is made bigger, changing this value to a large number will increase the amount of time interrupts are disabled during the load. It should also be kept small enough to avoid packet segmentation for the command protocal being used.

#### #define MM\_MISSION\_REV  0

##### Purpose:

Mission specific version number for MM application

##### Description:

An application version number consists of four parts: major version number, minor version number, revision number and mission specific revision number. The mission specific revision number is defined here and the other parts are defined in "mm\_version.h".

##### Limits:

Must be defined as a numeric value that is greater than or equal to zero.

#### #define MM\_OPT\_CODE\_MEM16\_MEMTYPE

##### Purpose:

Optional MEM16 compile switch

##### Description:

Compile switch to include code for the optional MM\_MEM16 memory. If defined the the code will be included. Otherwise it will be excluded.

##### Limits:

n/a

#### #define MM\_OPT\_CODE\_MEM32\_MEMTYPE

##### Purpose:

Optional MEM32 compile switch

##### Description:

Compile switch to include code for the optional MM\_MEM32 memory. If defined the code will be included. Otherwise the code will be excluded.

##### Limits:

n/a

#### #define MM\_OPT\_CODE\_MEM8\_MEMTYPE

##### Purpose:

Optional MEM8 compile switch

##### Description:

Compile switch to include code for the optional MM\_MEM8 memory. If defined the code will be included. Otherwise the code will be excluded.

##### Limits:

n/a

#### #define MM\_PROCESSOR\_CYCLE  0

##### Purpose:

Segment break processor delay

##### Description:

How many milliseconds to delay between segments for dump, load, and fill operations. A value of zero cycles through the OS scheduler, giving up what's left of the current timeslice.

##### Limits:

The MM app does not place a limit on this parameter. However, setting this value to a large number will increase the time required to process load, dump, and fill requests. It will also increase the likelyhood of MM being late responding to housekeeping requests since it cannot process such a request while a memory operation is in progress.

## fsw/src/mm\_app.c File Reference

#include "mm\_app.h"

#include "mm\_perfids.h"

#include "mm\_msgids.h"

#include "mm\_load.h"

#include "mm\_dump.h"

#include "mm\_utils.h"

#include "mm\_events.h"

#include "mm\_verify.h"

#include "mm\_version.h"

#include "mm\_platform\_cfg.h"

#include <string.h>

### Functions

* void [MM\_AppMain](#AAAAAAAABT) (void)

*CFS Memory Manager (MM) application entry point.*

* int32 [MM\_AppInit](#AAAAAAAABU) (void)

*Initialize the memory manager CFS application.*

* void [MM\_AppPipe](#AAAAAAAABV) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process a command pipe message.*

* void [MM\_HousekeepingCmd](#AAAAAAAABW) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process housekeeping request.*

* bool [MM\_NoopCmd](#AAAAAAAABX) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process noop command.*

* bool [MM\_ResetCmd](#AAAAAAAABY) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process reset counters command.*

* bool [MM\_LookupSymbolCmd](#AAAAAAAABZ) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process lookup symbol command.*

* bool [MM\_SymTblToFileCmd](#AAAAAAAACA) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Dump symbol table to file command.*

* bool [MM\_EepromWriteEnaCmd](#AAAAAAAACB) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Write-enable EEPROM command.*

* bool [MM\_EepromWriteDisCmd](#AAAAAAAACC) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Write-disable EEPROM command.*

### Variables

* [MM\_AppData\_t](#AAAAAAAACD) [MM\_AppData](#AAAAAAAACE)

### Function Documentation

#### int32 MM\_AppInit (void )

Initialize the memory manager CFS application.

##### Description

Memory manager application initialization routine. This function performs all the required startup steps to get the application registered with the cFE services so it can begin to receive command messages.

##### Assumptions, External Events, and Notes:

None

##### Returns:

#CFE\_SUCCESS Return codes from #CFE\_EVS\_Register Return codes from #CFE\_SB\_CreatePipe Return codes from #CFE\_SB\_Subscribe

#### void MM\_AppMain (void )

CFS Memory Manager (MM) application entry point.

##### Description

Memory Manager application entry point and main process loop.

##### Assumptions, External Events, and Notes:

None

#### void MM\_AppPipe (const CFE\_SB\_Buffer\_t \* *msg*)

Process a command pipe message.

##### Description

Processes a single software bus command pipe message. Checks the message and command IDs and calls the appropriate routine to handle the command.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *msg* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

#CFE\_SB\_RcvMsg

#### bool MM\_EepromWriteDisCmd (const CFE\_SB\_Buffer\_t \* *msg*)

Write-disable EEPROM command.

##### Description

Processes a EEPROM write disable ground command which calls the #CFE\_PSP\_EepromWriteDisable cFE function using the specified bank number.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *msg* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_DISABLE\_EEPROM\_WRITE\_CC](#AAAAAAAACF)

#### bool MM\_EepromWriteEnaCmd (const CFE\_SB\_Buffer\_t \* *msg*)

Write-enable EEPROM command.

##### Description

Processes a EEPROM write enable ground command which calls the #CFE\_PSP\_EepromWriteEnable cFE function using the specified bank number.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *msg* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_ENABLE\_EEPROM\_WRITE\_CC](#AAAAAAAACG)

#### void MM\_HousekeepingCmd (const CFE\_SB\_Buffer\_t \* *msg*)

Process housekeeping request.

##### Description

Processes an on-board housekeeping request message.

##### Assumptions, External Events, and Notes:

This command does not affect the command execution counter

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *msg* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

#### bool MM\_LookupSymbolCmd (const CFE\_SB\_Buffer\_t \* *msg*)

Process lookup symbol command.

##### Description

Processes a lookup symbol ground command which takes a symbol name and tries to resolve it to an address using the #OS\_SymbolLookup OSAL function.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *msg* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_LOOKUP\_SYM\_CC](#AAAAAAAACH)

#### bool MM\_NoopCmd (const CFE\_SB\_Buffer\_t \* *msg*)

Process noop command.

##### Description

Processes a noop ground command.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *msg* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_NOOP\_CC](#AAAAAAAACI)

#### bool MM\_ResetCmd (const CFE\_SB\_Buffer\_t \* *BufPtr*)

Process reset counters command.

##### Description

Processes a reset counters ground command which will reset the memory manager commmand error and command execution counters to zero.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *BufPtr* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_RESET\_CC](#AAAAAAAACJ)

#### bool MM\_SymTblToFileCmd (const CFE\_SB\_Buffer\_t \* *msg*)

Dump symbol table to file command.

##### Description

Processes a dump symbol table to file ground command which calls the #OS\_SymbolTableDump OSAL function using the specified dump file name.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *msg* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_SYMTBL\_TO\_FILE\_CC](#AAAAAAAACK)

### Variable Documentation

#### [MM\_AppData\_t](#AAAAAAAACD) MM\_AppData

#### 

## fsw/src/mm\_app.h File Reference

#include "mm\_msg.h"

#include "cfe.h"

### Data Structures

* struct [MM\_AppData\_t](#AAAAAAAACD)

### *MM global data structure.* Macros

* #define [MM\_MAX\_MEM\_TYPE\_STR\_LEN](#AAAAAAAACM)  11 /\* length of [MM\_EEPROM](#AAAAAAAACN) + \0 \*/

MM Command Pipe Parametera

#define [MM\_CMD\_PIPE\_DEPTH](#AAAAAAAACO)  (3 \* CFE\_PLATFORM\_SB\_DEFAULT\_MSG\_LIMIT)

#define [MM\_VERIFY\_DUMP](#AAAAAAAACP)  0

#define [MM\_VERIFY\_LOAD](#AAAAAAAACQ)  1

#define [MM\_VERIFY\_EVENT](#AAAAAAAACR)  2

#define [MM\_VERIFY\_FILL](#AAAAAAAACS)  3

#define [MM\_VERIFY\_WID](#AAAAAAAACT)  4

#define [MM\_VERIFY\_TYPES](#AAAAAAAACU)  5

### Functions

* void [MM\_AppMain](#AAAAAAAACV) (void)

*CFS Memory Manager (MM) application entry point.*

* int32 [MM\_AppInit](#AAAAAAAACW) (void)

*Initialize the memory manager CFS application.*

* void [MM\_AppPipe](#AAAAAAAACX) (const CFE\_SB\_Buffer\_t \*msg)

*Process a command pipe message.*

* void [MM\_HousekeepingCmd](#AAAAAAAACY) (const CFE\_SB\_Buffer\_t \*msg)

*Process housekeeping request.*

* bool [MM\_NoopCmd](#AAAAAAAACZ) (const CFE\_SB\_Buffer\_t \*msg)

*Process noop command.*

* bool [MM\_ResetCmd](#AAAAAAAADA) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process reset counters command.*

* bool [MM\_LookupSymbolCmd](#AAAAAAAADB) (const CFE\_SB\_Buffer\_t \*msg)

*Process lookup symbol command.*

* bool [MM\_SymTblToFileCmd](#AAAAAAAADC) (const CFE\_SB\_Buffer\_t \*msg)

*Dump symbol table to file command.*

* bool [MM\_EepromWriteEnaCmd](#AAAAAAAADD) (const CFE\_SB\_Buffer\_t \*msg)

*Write-enable EEPROM command.*

* bool [MM\_EepromWriteDisCmd](#AAAAAAAADE) (const CFE\_SB\_Buffer\_t \*msg)

*Write-disable EEPROM command.*

### Variables

* [MM\_AppData\_t](#AAAAAAAACD) [MM\_AppData](#AAAAAAAADF)

### Macro Definition Documentation

#### #define MM\_CMD\_PIPE\_DEPTH  (3 \* CFE\_PLATFORM\_SB\_DEFAULT\_MSG\_LIMIT)

#### #define MM\_MAX\_MEM\_TYPE\_STR\_LEN  11 /\* length of [MM\_EEPROM](#AAAAAAAACN) + \0 \*/

#### #define MM\_VERIFY\_DUMP  0

#### #define MM\_VERIFY\_EVENT  2

#### #define MM\_VERIFY\_FILL  3

#### #define MM\_VERIFY\_LOAD  1

#### #define MM\_VERIFY\_TYPES  5

#### #define MM\_VERIFY\_WID  4

### Function Documentation

#### int32 MM\_AppInit (void )

Initialize the memory manager CFS application.

##### Description

Memory manager application initialization routine. This function performs all the required startup steps to get the application registered with the cFE services so it can begin to receive command messages.

##### Assumptions, External Events, and Notes:

None

##### Returns:

#CFE\_SUCCESS Return codes from #CFE\_EVS\_Register Return codes from #CFE\_SB\_CreatePipe Return codes from #CFE\_SB\_Subscribe

#### void MM\_AppMain (void )

CFS Memory Manager (MM) application entry point.

##### Description

Memory Manager application entry point and main process loop.

##### Assumptions, External Events, and Notes:

None

#### void MM\_AppPipe (const CFE\_SB\_Buffer\_t \* *msg*)

Process a command pipe message.

##### Description

Processes a single software bus command pipe message. Checks the message and command IDs and calls the appropriate routine to handle the command.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *msg* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

#CFE\_SB\_RcvMsg

#### bool MM\_EepromWriteDisCmd (const CFE\_SB\_Buffer\_t \* *msg*)

Write-disable EEPROM command.

##### Description

Processes a EEPROM write disable ground command which calls the #CFE\_PSP\_EepromWriteDisable cFE function using the specified bank number.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *msg* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_DISABLE\_EEPROM\_WRITE\_CC](#AAAAAAAACF)

#### bool MM\_EepromWriteEnaCmd (const CFE\_SB\_Buffer\_t \* *msg*)

Write-enable EEPROM command.

##### Description

Processes a EEPROM write enable ground command which calls the #CFE\_PSP\_EepromWriteEnable cFE function using the specified bank number.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *msg* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_ENABLE\_EEPROM\_WRITE\_CC](#AAAAAAAACG)

#### void MM\_HousekeepingCmd (const CFE\_SB\_Buffer\_t \* *msg*)

Process housekeeping request.

##### Description

Processes an on-board housekeeping request message.

##### Assumptions, External Events, and Notes:

This command does not affect the command execution counter

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *msg* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

#### bool MM\_LookupSymbolCmd (const CFE\_SB\_Buffer\_t \* *msg*)

Process lookup symbol command.

##### Description

Processes a lookup symbol ground command which takes a symbol name and tries to resolve it to an address using the #OS\_SymbolLookup OSAL function.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *msg* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_LOOKUP\_SYM\_CC](#AAAAAAAACH)

#### bool MM\_NoopCmd (const CFE\_SB\_Buffer\_t \* *msg*)

Process noop command.

##### Description

Processes a noop ground command.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *msg* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_NOOP\_CC](#AAAAAAAACI)

#### bool MM\_ResetCmd (const CFE\_SB\_Buffer\_t \* *BufPtr*)

Process reset counters command.

##### Description

Processes a reset counters ground command which will reset the memory manager commmand error and command execution counters to zero.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *BufPtr* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_RESET\_CC](#AAAAAAAACJ)

#### bool MM\_SymTblToFileCmd (const CFE\_SB\_Buffer\_t \* *msg*)

Dump symbol table to file command.

##### Description

Processes a dump symbol table to file ground command which calls the #OS\_SymbolTableDump OSAL function using the specified dump file name.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *msg* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_SYMTBL\_TO\_FILE\_CC](#AAAAAAAACK)

### Variable Documentation

#### [MM\_AppData\_t](#AAAAAAAACD) MM\_AppData

#### 

## fsw/src/mm\_dump.c File Reference

#include "mm\_app.h"

#include "mm\_dump.h"

#include "mm\_events.h"

#include "mm\_mem32.h"

#include "mm\_mem16.h"

#include "mm\_mem8.h"

#include "mm\_utils.h"

#include "mm\_mission\_cfg.h"

#include "cfs\_utils.h"

#include <string.h>

### Functions

* bool [MM\_PeekCmd](#AAAAAAAADH) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process memory peek command.*

* bool [MM\_PeekMem](#AAAAAAAADI) (const [MM\_PeekCmd\_t](#AAAAAAAADJ) \*CmdPtr, cpuaddr SrcAddress)

*Memory peek.*

* bool [MM\_DumpMemToFileCmd](#AAAAAAAADK) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process memory dump to file command.*

* bool [MM\_DumpMemToFile](#AAAAAAAADL) (uint32 FileHandle, const char \*FileName, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \*FileHeader)

*Memory dump to file.*

* bool [MM\_WriteFileHeaders](#AAAAAAAADN) (const char \*FileName, int32 FileHandle, CFE\_FS\_Header\_t \*CFEHeader, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \*MMHeader)

*Write the cFE primary and and MM secondary file headers.*

* bool [MM\_DumpInEventCmd](#AAAAAAAADO) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process memory dump in event command.*

* bool [MM\_FillDumpInEventBuffer](#AAAAAAAADP) (cpuaddr SrcAddress, const [MM\_DumpInEventCmd\_t](#AAAAAAAADQ) \*CmdPtr, void \*DumpBuffer)

*Fill dump memory in event message buffer.*

### Variables

* [MM\_AppData\_t](#AAAAAAAACD) [MM\_AppData](#AAAAAAAADR)

### Function Documentation

#### bool MM\_DumpInEventCmd (const CFE\_SB\_Buffer\_t \* *BufPtr*)

Process memory dump in event command.

##### Description

Processes the memory dump in event command that will read up to [MM\_MAX\_DUMP\_INEVENT\_BYTES](#AAAAAAAADS) from memory and report the data in an event message.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *BufPtr* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_DUMP\_IN\_EVENT\_CC](#AAAAAAAADT), [MM\_MAX\_DUMP\_INEVENT\_BYTES](#AAAAAAAADS)

#### bool MM\_DumpMemToFile (uint32 *FileHandle*, const char \* *FileName*, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \* *FileHeader*)

Memory dump to file.

##### Description

Support function for [MM\_DumpMemToFileCmd](#AAAAAAAADU). This routine will read an address range and store the data in a file.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *FileHandle* | The open file handle of the dump file |
| in | *FileName* | A pointer to a character string holding the dump file name |
| in | *FileHeader* | A [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) pointer to the dump file header structure initialized with data based upon the command message parameters |

##### Returns:

Returns TRUE if the dump completed successfully Returns FALSE if the dump failed due to an error

#### bool MM\_DumpMemToFileCmd (const CFE\_SB\_Buffer\_t \* *BufPtr*)

Process memory dump to file command.

##### Description

Processes the memory dump to file command that will read a address range of memory and store the data in a command specified file.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *BufPtr* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_DUMP\_MEM\_TO\_FILE\_CC](#AAAAAAAADV)

#### bool MM\_FillDumpInEventBuffer (cpuaddr *SrcAddress*, const [MM\_DumpInEventCmd\_t](#AAAAAAAADQ) \* *CmdPtr*, void \* *DumpBuffer*)

Fill dump memory in event message buffer.

##### Description

Support function for [MM\_DumpInEventCmd](#AAAAAAAADW). This routine will read an address range and store the data in a byte array. It will properly adjust for optional memory types that may require 16 or 32 bit reads.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *SrcAddress* | The source address to read from |
| in | *CmdPtr* | A [MM\_DumpInEventCmd\_t](#AAAAAAAADQ) pointer to the dump in event command message |
| in | *DumpBuffer* | A pointer to the byte array to store the dump data in |
| out | *\*DumpBuffer* | A pointer to the byte array holding the dump data |

##### Returns:

Returns TRUE if all PSP memory access functions succeed Returns FALSE if any PSP memory access function fails

#### bool MM\_PeekCmd (const CFE\_SB\_Buffer\_t \* *BufPtr*)

Process memory peek command.

##### Description

Processes the memory peek command that will read a memory location and report the data in an event message.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *BufPtr* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_PEEK\_CC](#AAAAAAAADX)

#### bool MM\_PeekMem (const [MM\_PeekCmd\_t](#AAAAAAAADJ) \* *CmdPtr*, cpuaddr *SrcAddress*)

Memory peek.

##### Description

Support function for [MM\_PeekCmd](#AAAAAAAADY). This routine will read 8, 16, or 32 bits of data and send it in an event message.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *CmdPtr* | A [MM\_PeekCmd\_t](#AAAAAAAADJ) pointer to the peek command message |
| in | *SrcAddress* | The source address for the peek operation |

#### bool MM\_WriteFileHeaders (const char \* *FileName*, int32 *FileHandle*, CFE\_FS\_Header\_t \* *CFEHeader*, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \* *MMHeader*)

Write the cFE primary and and MM secondary file headers.

##### Description

Support function for [MM\_DumpMemToFileCmd](#AAAAAAAADU). This routine will write the cFE primary and MM secondary headers to the file specified by the FileHandle.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *FileName* | A pointer to a character string holding the file name (used only for error event messages). |
| in | *FileHandle* | File Descriptor obtained from a previous call to #OS\_open that is associated with the file whose headers are to be written. |
| in | *CFEHeader* | A #CFE\_FS\_Header\_t pointer to the cFE primary file header structure to be written. |
| in | *MMHeader* | A [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) pointer to the MM secondary file header structure to be written. |

##### Returns:

Returns TRUE if the headers were written successfully Returns FALSE if a write error occurred

### Variable Documentation

#### [MM\_AppData\_t](#AAAAAAAACD) MM\_AppData

#### 

## fsw/src/mm\_dump.h File Reference

#include "cfe.h"

#include "mm\_msg.h"

#include "mm\_filedefs.h"

### Macros

Maximum dump bytes in an event string#define [MM\_MAX\_DUMP\_INEVENT\_BYTES](#AAAAAAAADS)  ((CFE\_MISSION\_EVS\_MAX\_MESSAGE\_LENGTH - (13 + 33)) / 5)

Dump in an event scratch string size#define [MM\_DUMPINEVENT\_TEMP\_CHARS](#AAAAAAAAEA)  36

### Functions

* bool [MM\_PeekMem](#AAAAAAAAEB) (const [MM\_PeekCmd\_t](#AAAAAAAADJ) \*CmdPtr, cpuaddr SrcAddress)

*Memory peek.*

* bool [MM\_DumpMemToFile](#AAAAAAAAEC) (uint32 FileHandle, const char \*FileName, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \*FileHeader)

*Memory dump to file.*

* bool [MM\_WriteFileHeaders](#AAAAAAAAED) (const char \*FileName, int32 FileHandle, CFE\_FS\_Header\_t \*CFEHeader, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \*MMHeader)

*Write the cFE primary and and MM secondary file headers.*

* bool [MM\_FillDumpInEventBuffer](#AAAAAAAAEE) (cpuaddr SrcAddress, const [MM\_DumpInEventCmd\_t](#AAAAAAAADQ) \*CmdPtr, void \*DumpBuffer)

*Fill dump memory in event message buffer.*

* bool [MM\_PeekCmd](#AAAAAAAADY) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process memory peek command.*

* bool [MM\_DumpMemToFileCmd](#AAAAAAAADU) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process memory dump to file command.*

* bool [MM\_DumpInEventCmd](#AAAAAAAADW) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process memory dump in event command.*

### Macro Definition Documentation

#### #define MM\_DUMPINEVENT\_TEMP\_CHARS  36

#### #define MM\_MAX\_DUMP\_INEVENT\_BYTES  ((CFE\_MISSION\_EVS\_MAX\_MESSAGE\_LENGTH - (13 + 33)) / 5)

### Function Documentation

#### bool MM\_DumpInEventCmd (const CFE\_SB\_Buffer\_t \* *BufPtr*)

Process memory dump in event command.

##### Description

Processes the memory dump in event command that will read up to [MM\_MAX\_DUMP\_INEVENT\_BYTES](#AAAAAAAADS) from memory and report the data in an event message.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *BufPtr* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_DUMP\_IN\_EVENT\_CC](#AAAAAAAADT), [MM\_MAX\_DUMP\_INEVENT\_BYTES](#AAAAAAAADS)

#### bool MM\_DumpMemToFile (uint32 *FileHandle*, const char \* *FileName*, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \* *FileHeader*)

Memory dump to file.

##### Description

Support function for [MM\_DumpMemToFileCmd](#AAAAAAAADU). This routine will read an address range and store the data in a file.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *FileHandle* | The open file handle of the dump file |
| in | *FileName* | A pointer to a character string holding the dump file name |
| in | *FileHeader* | A [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) pointer to the dump file header structure initialized with data based upon the command message parameters |

##### Returns:

Returns TRUE if the dump completed successfully Returns FALSE if the dump failed due to an error

#### bool MM\_DumpMemToFileCmd (const CFE\_SB\_Buffer\_t \* *BufPtr*)

Process memory dump to file command.

##### Description

Processes the memory dump to file command that will read a address range of memory and store the data in a command specified file.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *BufPtr* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_DUMP\_MEM\_TO\_FILE\_CC](#AAAAAAAADV)

#### bool MM\_FillDumpInEventBuffer (cpuaddr *SrcAddress*, const [MM\_DumpInEventCmd\_t](#AAAAAAAADQ) \* *CmdPtr*, void \* *DumpBuffer*)

Fill dump memory in event message buffer.

##### Description

Support function for [MM\_DumpInEventCmd](#AAAAAAAADW). This routine will read an address range and store the data in a byte array. It will properly adjust for optional memory types that may require 16 or 32 bit reads.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *SrcAddress* | The source address to read from |
| in | *CmdPtr* | A [MM\_DumpInEventCmd\_t](#AAAAAAAADQ) pointer to the dump in event command message |
| in | *DumpBuffer* | A pointer to the byte array to store the dump data in |
| out | *\*DumpBuffer* | A pointer to the byte array holding the dump data |

##### Returns:

Returns TRUE if all PSP memory access functions succeed Returns FALSE if any PSP memory access function fails

#### bool MM\_PeekCmd (const CFE\_SB\_Buffer\_t \* *BufPtr*)

Process memory peek command.

##### Description

Processes the memory peek command that will read a memory location and report the data in an event message.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *BufPtr* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_PEEK\_CC](#AAAAAAAADX)

#### bool MM\_PeekMem (const [MM\_PeekCmd\_t](#AAAAAAAADJ) \* *CmdPtr*, cpuaddr *SrcAddress*)

Memory peek.

##### Description

Support function for [MM\_PeekCmd](#AAAAAAAADY). This routine will read 8, 16, or 32 bits of data and send it in an event message.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *CmdPtr* | A [MM\_PeekCmd\_t](#AAAAAAAADJ) pointer to the peek command message |
| in | *SrcAddress* | The source address for the peek operation |

#### bool MM\_WriteFileHeaders (const char \* *FileName*, int32 *FileHandle*, CFE\_FS\_Header\_t \* *CFEHeader*, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \* *MMHeader*)

Write the cFE primary and and MM secondary file headers.

##### Description

Support function for [MM\_DumpMemToFileCmd](#AAAAAAAADU). This routine will write the cFE primary and MM secondary headers to the file specified by the FileHandle.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *FileName* | A pointer to a character string holding the file name (used only for error event messages). |
| in | *FileHandle* | File Descriptor obtained from a previous call to #OS\_open that is associated with the file whose headers are to be written. |
| in | *CFEHeader* | A #CFE\_FS\_Header\_t pointer to the cFE primary file header structure to be written. |
| in | *MMHeader* | A [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) pointer to the MM secondary file header structure to be written. |

##### Returns:

Returns TRUE if the headers were written successfully Returns FALSE if a write error occurred

## fsw/src/mm\_events.h File Reference

### Macros

* #define [MM\_INIT\_INF\_EID](#AAAAAAAAEG)  1

*'MM Initialized. Version %d.%d.%d.%d'*

* #define [MM\_NOOP\_INF\_EID](#AAAAAAAAEH)  2

*'No-op command. Version %d.%d.%d.%d'*

* #define [MM\_RESET\_INF\_EID](#AAAAAAAAEI)  3

*'Reset counters command received'*

* #define [MM\_LOAD\_WID\_INF\_EID](#AAAAAAAAEJ)  4

*'Load Memory WID Command: Wrote %d bytes to address: 0x%08X'*

* #define [MM\_LD\_MEM\_FILE\_INF\_EID](#AAAAAAAAEK)  5

*'Load Memory From File Command: Loaded %d bytes to address 0x%08X from file '%s''*

* #define [MM\_FILL\_INF\_EID](#AAAAAAAAEL)  6

*'Fill Memory Command: Filled %d bytes at address: 0x%08X with pattern: 0x%08X'*

* #define [MM\_PEEK\_BYTE\_INF\_EID](#AAAAAAAAEM)  7

*'Peek Command: Addr = 0x%08X Size = 8 bits Data = 0x%02X'*

* #define [MM\_PEEK\_WORD\_INF\_EID](#AAAAAAAAEN)  8

*'Peek Command: Addr = 0x%08X Size = 16 bits Data = 0x%04X'*

* #define [MM\_PEEK\_DWORD\_INF\_EID](#AAAAAAAAEO)  9

*'Peek Command: Addr = 0x%08X Size = 32 bits Data = 0x%08X'*

* #define [MM\_POKE\_BYTE\_INF\_EID](#AAAAAAAAEP)  10

*'Poke Command: Addr = 0x%08X Size = 8 bits Data = 0x%02X'*

* #define [MM\_POKE\_WORD\_INF\_EID](#AAAAAAAAEQ)  11

*'Poke Command: Addr = 0x%08X Size = 16 bits Data = 0x%04X'*

* #define [MM\_POKE\_DWORD\_INF\_EID](#AAAAAAAAER)  12

*'Poke Command: Addr = 0x%08X Size = 32 bits Data = 0x%08X'*

* #define [MM\_DMP\_MEM\_FILE\_INF\_EID](#AAAAAAAAES)  13

*'Dump Memory To File Command: Dumped %d bytes from address 0x%08X to file '%s''*

* #define [MM\_DUMP\_INEVENT\_INF\_EID](#AAAAAAAAET)  14

*'Memory Dump: 0x%02X from address: 0x%08lX'*

* #define [MM\_PIPE\_ERR\_EID](#AAAAAAAAEU)  15

*'SB Pipe Read Error, App will exit. RC = 0x%08X'*

* #define [MM\_MID\_ERR\_EID](#AAAAAAAAEV)  16

*'Invalid command pipe message ID: 0x%X'*

* #define [MM\_CC1\_ERR\_EID](#AAAAAAAAEW)  17

*'Invalid ground command code: ID = 0x%X, CC = %d'*

* #define [MM\_LEN\_ERR\_EID](#AAAAAAAAEX)  18

*'Invalid msg length: ID = 0x%04X, CC = %d, Len = %d, Expected = %d'*

* #define [MM\_MEMTYPE\_ERR\_EID](#AAAAAAAAEY)  19

*'Invalid memory type specified: MemType = %d'*

* #define [MM\_SYMNAME\_ERR\_EID](#AAAAAAAAEZ)  20

*'Symbolic address can't be resolved: Name = '%s''*

* #define [MM\_DATA\_SIZE\_BYTES\_ERR\_EID](#AAAAAAAAFA)  21

*'Data size in bytes invalid or exceeds limits: Data Size = %d'*

* #define [MM\_DATA\_SIZE\_BITS\_ERR\_EID](#AAAAAAAAFB)  22

*'Data size in bits invalid: Data Size = %d'*

* #define [MM\_ALIGN32\_ERR\_EID](#AAAAAAAAFC)  23

*'Data and address not 32 bit aligned: Addr = 0x%08X Size = %d'*

* #define [MM\_ALIGN16\_ERR\_EID](#AAAAAAAAFD)  24

*'Data and address not 16 bit aligned: Addr = 0x%08X Size = %d'*

* #define [MM\_OS\_MEMVALIDATE\_ERR\_EID](#AAAAAAAAFE)  25

*'CFE\_PSP\_MemValidate error received: RC = 0x%08X Addr = 0x%08X Size = %d MemType = %d'*

* #define [MM\_LOAD\_FILE\_CRC\_ERR\_EID](#AAAAAAAAFF)  26

*'Load file CRC failure: Expected = 0x%X Calculated = 0x%X File = '%s''*

* #define [MM\_LOAD\_WID\_CRC\_ERR\_EID](#AAAAAAAAFG)  27

*'Interrupts Disabled Load CRC failure: Expected = 0x%X Calculated = 0x%X'*

* #define [MM\_OS\_EEPROMWRITE8\_ERR\_EID](#AAAAAAAAFH)  28

*'OS\_EepromWrite8 error received: RC = 0x%08X Addr = 0x%08X'*

* #define [MM\_OS\_EEPROMWRITE16\_ERR\_EID](#AAAAAAAAFI)  29

*'OS\_EepromWrite16 error received: RC = 0x%08X Addr = 0x%08X'*

* #define [MM\_OS\_EEPROMWRITE32\_ERR\_EID](#AAAAAAAAFJ)  30

*'OS\_EepromWrite32 error received: RC = 0x%08X Addr = 0x%08X'*

* #define [MM\_OS\_CREAT\_ERR\_EID](#AAAAAAAAFK)  31

*'OS\_OpenCreate error received: RC = 0x%08X File = '%s''*

* #define [MM\_OS\_OPEN\_ERR\_EID](#AAAAAAAAFL)  32

*'OS\_OpenCreate error received: RC = 0x%08X File = '%s''*

* #define [MM\_OS\_CLOSE\_ERR\_EID](#AAAAAAAAFM)  33

*'OS\_close error received: RC = 0x%08X File = '%s''*

* #define [MM\_OS\_READ\_ERR\_EID](#AAAAAAAAFN)  34

*'OS\_read error received: RC = 0x%08X File = '%s''*

* #define [MM\_OS\_READ\_EXP\_ERR\_EID](#AAAAAAAAFO)  35

*'OS\_read error received: RC = 0x%08X Expected = %d File = '%s''*

* #define [MM\_OS\_WRITE\_EXP\_ERR\_EID](#AAAAAAAAFP)  36

*'OS\_write error received: RC = 0x%08X Expected = %d File = '%s''*

* #define [MM\_OS\_STAT\_ERR\_EID](#AAAAAAAAFQ)  37

*'OS\_stat error received: RC = 0x%08X File = '%s''*

* #define [MM\_CFS\_COMPUTECRCFROMFILE\_ERR\_EID](#AAAAAAAAFR)  38

*'CFS\_ComputeCRCFromFile error received: RC = 0x%08X File = '%s''*

* #define [MM\_CMD\_FNAME\_ERR\_EID](#AAAAAAAAFS)  39

*'Command specified filename invalid: Name = '%s''*

* #define [MM\_LD\_FILE\_SIZE\_ERR\_EID](#AAAAAAAAFT)  40

*'Load file size error: Reported by OS = %d Expected = %d File = '%s''*

* #define [MM\_FILE\_LOAD\_PARAMS\_ERR\_EID](#AAAAAAAAFU)  41

*'Load file failed parameters check: File = '%s''*

* #define [MM\_CFE\_FS\_READHDR\_ERR\_EID](#AAAAAAAAFV)  42

*'CFE\_FS\_ReadHeader error received: RC = 0x%08X Expected = %d File = '%s''*

* #define [MM\_CFE\_FS\_WRITEHDR\_ERR\_EID](#AAAAAAAAFW)  43

*'CFE\_FS\_WriteHeader error received: RC = 0x%08X Expected = %d File = '%s''*

* #define [MM\_HKREQ\_LEN\_ERR\_EID](#AAAAAAAAFX)  44

*'Invalid HK request msg length: ID = 0x%04X, CC = %d, Len = %d, Expected = %d'*

* #define [MM\_SYM\_LOOKUP\_INF\_EID](#AAAAAAAAFY)  45

*'Symbol Lookup Command: Name = '%s' Addr = 0x%08X'*

* #define [MM\_SYMNAME\_NUL\_ERR\_EID](#AAAAAAAAFZ)  46

*'NUL (empty) string specified as symbol name'*

* #define [MM\_SYMTBL\_TO\_FILE\_INF\_EID](#AAAAAAAAGA)  47

*'Symbol Table Dump to File Started: Name = '%s''*

* #define [MM\_SYMFILENAME\_NUL\_ERR\_EID](#AAAAAAAAGB)  48

*'NUL (empty) string specified as symbol dump file name'*

* #define [MM\_SYMTBL\_TO\_FILE\_FAIL\_ERR\_EID](#AAAAAAAAGC)  49

*'Error dumping symbol table, OS\_Status= 0xX, File='s''*

* #define [MM\_SYMTBL\_TO\_FILE\_INVALID\_ERR\_EID](#AAAAAAAAGD)  50

*'Illegal characters in target filename, File='s''*

* #define [MM\_EEPROM\_WRITE\_ENA\_INF\_EID](#AAAAAAAAGE)  51

*'EEPROM bank d write enabled, cFE\_Status= 0xX'*

* #define [MM\_EEPROM\_WRITE\_ENA\_ERR\_EID](#AAAAAAAAGF)  52

*'Error requesting EEPROM bank d write enable, cFE\_Status= 0xX'*

* #define [MM\_EEPROM\_WRITE\_DIS\_INF\_EID](#AAAAAAAAGG)  53

*'EEPROM bank d write disabled, cFE\_Status= 0xX'*

* #define [MM\_EEPROM\_WRITE\_DIS\_ERR\_EID](#AAAAAAAAGH)  54

*'Error requesting EEPROM bank d write disable, cFE\_Status= 0xX'*

* #define [MM\_OS\_ZERO\_READ\_ERR\_EID](#AAAAAAAAGI)  55

*'Zero bytes read by OS\_read of file '%s''*

* #define [MM\_PSP\_READ\_ERR\_EID](#AAAAAAAAGJ)  56

*'PSP read memory error: RC=0x%08X, Src=0x%08X, Tgt=0x%08X, Type='s''*

* #define [MM\_PSP\_WRITE\_ERR\_EID](#AAAAAAAAGK)  57

*'PSP write memory error: RC=0x%08X, Address=0x%08X, MemType='s''*

* #define [MM\_CR\_PIPE\_ERR\_EID](#AAAAAAAAGL)  60

*'Error Creating SB Pipe, RC = 0x%08X'*

* #define [MM\_HK\_SUB\_ERR\_EID](#AAAAAAAAGM)  61

*'Error Subscribing to HK Request, RC = 0x%08X'*

* #define [MM\_CMD\_SUB\_ERR\_EID](#AAAAAAAAGN)  62

*'Error Subscribing to MM Command, RC = 0x%08X'*

* #define [MM\_FILL\_MEM32\_ALIGN\_WARN\_INF\_EID](#AAAAAAAAGO)  63

*'MM\_FillMem32 NumOfBytes not multiple of 4. Reducing from d to d.'*

* #define [MM\_FILL\_MEM16\_ALIGN\_WARN\_INF\_EID](#AAAAAAAAGP)  64

*'MM\_FillMem16 NumOfBytes not multiple of 2. Reducing from d to d.'*

### Macro Definition Documentation

#### #define MM\_ALIGN16\_ERR\_EID  24

'Data and address not 16 bit aligned: Addr = 0x%08X Size = %d'

##### Event Message:

'Data and address not 16 bit aligned: Addr = 0x%08X Size = %d'

##### Type: ERROR

##### Cause:

This event message is issued when command execution requires 16 bit wide memory access and the data size and address specified are not both 16 bit aligned.

The Addr field is the address and the Size field is the specified size in bytes, that failed the alignment check.

#### #define MM\_ALIGN32\_ERR\_EID  23

'Data and address not 32 bit aligned: Addr = 0x%08X Size = %d'

##### Event Message:

'Data and address not 32 bit aligned: Addr = 0x%08X Size = %d'

##### Type: ERROR

##### Cause:

This event message is issued when command execution requires 32 bit wide memory access and the data size and address specified are not both 32 bit aligned.

The Addr field is the address and the Size field is the specified size in bytes, that failed the alignment check.

#### #define MM\_CC1\_ERR\_EID  17

'Invalid ground command code: ID = 0x%X, CC = %d'

##### Event Message:

'Invalid ground command code: ID = 0x%X, CC = %d'

##### Type: ERROR

##### Cause:

This event message is issued when a software bus message is received with an invalid command code.

The ID field contains the message ID, the CC field contains the command code that generated the error.

#### #define MM\_CFE\_FS\_READHDR\_ERR\_EID  42

'CFE\_FS\_ReadHeader error received: RC = 0x%08X Expected = %d File = '%s''

##### Event Message:

'CFE\_FS\_ReadHeader error received: RC = 0x%08X Expected = %d File = '%s''

##### Type: ERROR

##### Cause:

This event message is issued when a call to the #CFE\_FS\_ReadHeader function returns some value other than the expected number of bytes read.

The RC field is the return code, the Expected field is the expected return value and the File is the filename from the #CFE\_FS\_ReadHeader call that generated the error.

#### #define MM\_CFE\_FS\_WRITEHDR\_ERR\_EID  43

'CFE\_FS\_WriteHeader error received: RC = 0x%08X Expected = %d File = '%s''

##### Event Message:

'CFE\_FS\_WriteHeader error received: RC = 0x%08X Expected = %d File = '%s''

##### Type: ERROR

##### Cause:

This event message is issued when a call to the #CFE\_FS\_WriteHeader function returns some value other than the expected number of bytes written.

The RC field is the return code, the Expected field is the expected return value and the File is the filename from the #CFE\_FS\_WriteHeader call that generated the error.

#### #define MM\_CFS\_COMPUTECRCFROMFILE\_ERR\_EID  38

'CFS\_ComputeCRCFromFile error received: RC = 0x%08X File = '%s''

##### Event Message:

'CFS\_ComputeCRCFromFile error received: RC = 0x%08X File = '%s''

##### Type: ERROR

##### Cause:

This event message is issued when a call to the #CFS\_ComputeCRCFromFile function returns some value other than #OS\_SUCCESS.

The RC field is the return code and File is the filename from the #CFS\_ComputeCRCFromFile call that generated the error.

#### #define MM\_CMD\_FNAME\_ERR\_EID  39

'Command specified filename invalid: Name = '%s''

##### Event Message:

'Command specified filename invalid: Name = '%s''

##### Type: ERROR

##### Cause:

This event message is issued when a filename included in a command message fails a check for prohibited characters

The Name field holds the filename string that generated the error.

#### #define MM\_CMD\_SUB\_ERR\_EID  62

'Error Subscribing to MM Command, RC = 0x%08X'

##### Event Message:

'Error Subscribing to MM Command, RC = 0x%08X'

##### Type: ERROR

##### Cause:

This event message is issued when a call to #CFE\_SB\_Subscribe returns something other than CFE\_SUCCESS when called for MM\_CMD\_MID.

The RC field is the function return code.

#### #define MM\_CR\_PIPE\_ERR\_EID  60

'Error Creating SB Pipe, RC = 0x%08X'

##### Event Message:

'Error Creating SB Pipe, RC = 0x%08X'

##### Type: ERROR

##### Cause:

This event message is issued when a call to #CFE\_SB\_CreatePipe returns something other than CFE\_SUCCESS.

The RC field is the function return code.

#### #define MM\_DATA\_SIZE\_BITS\_ERR\_EID  22

'Data size in bits invalid: Data Size = %d'

##### Event Message:

'Data size in bits invalid: Data Size = %d'

##### Type: ERROR

##### Cause:

This event message is issued when a command specified bit width for a peek or poke operation is either undefined or not valid for the specified memory type.

The Data Size field is the data size in bits that generated the error.

#### #define MM\_DATA\_SIZE\_BYTES\_ERR\_EID  21

'Data size in bytes invalid or exceeds limits: Data Size = %d'

##### Event Message:

'Data size in bytes invalid or exceeds limits: Data Size = %d'

##### Type: ERROR

##### Cause:

This event message is issued when a command or load file specifies a number of bytes that is either zero or exceeds the limits specified by the MM configuration parameters

The Data Size field is the data size in bytes that generated the error.

#### #define MM\_DMP\_MEM\_FILE\_INF\_EID  13

'Dump Memory To File Command: Dumped %d bytes from address 0x%08X to file '%s''

##### Event Message:

'Dump Memory To File Command: Dumped %d bytes from address 0x%08X to file '%s''

##### Type: INFORMATIONAL

##### Cause:

This event message is issued when a dump memory to file command has been executed.

The bytes field identifies how many bytes were read, the address field shows the fully resolved source address of the dump, the file field identifies the name of the file used for the dump.

#### #define MM\_DUMP\_INEVENT\_INF\_EID  14

'Memory Dump: 0x%02X from address: 0x%08lX'

##### Event Message:

'Memory Dump: 0x%02X from address: 0x%08lX'

##### Type: INFORMATIONAL

##### Cause:

This event message is issued in response to a dump memory in event message command.

The 0x%02X field is a single byte of data and will be repeated according to the requested number of dump bytes, the address field shows the fully resolved source address of the dump.

#### #define MM\_EEPROM\_WRITE\_DIS\_ERR\_EID  54

'Error requesting EEPROM bank d write disable, cFE\_Status= 0xX'

##### Event Message:

'Error requesting EEPROM bank d write disable, cFE\_Status= 0xX'

##### Type: ERROR

##### Cause:

This event message is issued when a request to disable writing to a specified EEPROM bank results in an error status from the PSP.

The bank field identifies the requested EEPROM bank to be write-disabled.

The CFE\_Status field holds the return code from the call to #CFE\_PSP\_EepromWriteDisable

#### #define MM\_EEPROM\_WRITE\_DIS\_INF\_EID  53

'EEPROM bank d write disabled, cFE\_Status= 0xX'

##### Event Message:

'EEPROM bank d write disabled, cFE\_Status= 0xX'

##### Type: INFORMATIONAL

##### Cause:

This event message is issued when a request to disable writing to a specified EEPROM bank results in a success status from the PSP.

The bank field identifies the requested EEPROM bank to be write-disabled.

The CFE\_Status field holds the return code from the call to #CFE\_PSP\_EepromWriteDisable

#### #define MM\_EEPROM\_WRITE\_ENA\_ERR\_EID  52

'Error requesting EEPROM bank d write enable, cFE\_Status= 0xX'

##### Event Message:

'Error requesting EEPROM bank d write enable, cFE\_Status= 0xX'

##### Type: ERROR

##### Cause:

This event message is issued when a request to enable writing to a specified EEPROM bank results in an error status from the PSP.

The bank field identifies the requested EEPROM bank to be write-enabled.

The CFE\_Status field holds the return code from the call to #CFE\_PSP\_EepromWriteEnable

#### #define MM\_EEPROM\_WRITE\_ENA\_INF\_EID  51

'EEPROM bank d write enabled, cFE\_Status= 0xX'

##### Event Message:

'EEPROM bank d write enabled, cFE\_Status= 0xX'

##### Type: INFORMATIONAL

##### Cause:

This event message is issued when a request to enable writing to a specified EEPROM bank results in a success status from the PSP.

The bank field identifies the requested EEPROM bank to be write-enabled The CFE\_Status field holds the return code from the call to #CFE\_PSP\_EepromWriteEnable

#### #define MM\_FILE\_LOAD\_PARAMS\_ERR\_EID  41

'Load file failed parameters check: File = '%s''

##### Event Message:

'Load file failed parameters check: File = '%s''

##### Type: ERROR

##### Cause:

This event message is issued when a load file fails one of a series of parameter checks on the Destination Address, Memory Type, and Byte Size specified in the load file header. Another error event will be issued with the specific error, this is a supplemental message that echos the name of the file that failed.

The File field holds the name of the file that failed the parameter checks.

#### #define MM\_FILL\_INF\_EID  6

'Fill Memory Command: Filled %d bytes at address: 0x%08X with pattern: 0x%08X'

##### Event Message:

'Fill Memory Command: Filled %d bytes at address: 0x%08X with pattern: 0x%08X'

##### Type: INFORMATIONAL

##### Cause:

This event message is issued when a fill memory command has been executed.

The bytes field identifies how many bytes were written, the address field shows the fully resolved destination address of the fill, the pattern field identifies the fill pattern used.

#### #define MM\_FILL\_MEM16\_ALIGN\_WARN\_INF\_EID  64

'MM\_FillMem16 NumOfBytes not multiple of 2. Reducing from d to d.'

##### Event Message:

'MM\_FillMem16 NumOfBytes not multiple of 2. Reducing from d to d.'

##### Type: INFORMATION

##### Cause:

This event message is issued when a call to [MM\_FillMem16](#AAAAAAAAGQ) is called with a NumOfBytes value that is not divisible by 2.

The from field is the original NumOfBytes parameter. The to field is the rounded down NumOfBytes value

#### #define MM\_FILL\_MEM32\_ALIGN\_WARN\_INF\_EID  63

'MM\_FillMem32 NumOfBytes not multiple of 4. Reducing from d to d.'

##### Event Message:

'MM\_FillMem32 NumOfBytes not multiple of 4. Reducing from d to d.'

##### Type: INFORMATION

##### Cause:

This event message is issued when a call to [MM\_FillMem32](#AAAAAAAAGR) is called with a NumOfBytes value that is not divisible by 4.

The from field is the original NumOfBytes parameter. The to field is the rounded down NumOfBytes value

#### #define MM\_HK\_SUB\_ERR\_EID  61

'Error Subscribing to HK Request, RC = 0x%08X'

##### Event Message:

'Error Subscribing to HK Request, RC = 0x%08X'

##### Type: ERROR

##### Cause:

This event message is issued when a call to #CFE\_SB\_Subscribe returns something other than CFE\_SUCCESS when called for MM\_SEND\_HK\_MID.

The RC field is the function return code.

#### #define MM\_HKREQ\_LEN\_ERR\_EID  44

'Invalid HK request msg length: ID = 0x%04X, CC = %d, Len = %d, Expected = %d'

##### Event Message:

'Invalid HK request msg length: ID = 0x%04X, CC = %d, Len = %d, Expected = %d'

##### Type: ERROR

##### Cause:

This event message is issued when housekeeping request message is received with a message length that doesn't match the expected value.

The ID field contains the message ID, the CC field contains the command code, the Len field is the actual length returned by the CFE\_SB\_GetTotalMsgLength call, and the Expected field is the expected length.

#### #define MM\_INIT\_INF\_EID  1

'MM Initialized. Version %d.%d.%d.%d'

##### Event Message:

'MM Initialized. Version %d.%d.%d.%d'

##### Type: INFORMATIONAL

##### Cause:

This event message is issued when the CFS Memory Manager has completed initialization.

The Version fields contain the [MM\_MAJOR\_VERSION](#AAAAAAAAGS), [MM\_MINOR\_VERSION](#AAAAAAAAGT), [MM\_REVISION](#AAAAAAAAGU), and [MM\_MISSION\_REV](#AAAAAAAABR) version identifiers.

#### #define MM\_LD\_FILE\_SIZE\_ERR\_EID  40

'Load file size error: Reported by OS = %d Expected = %d File = '%s''

##### Event Message:

'Load file size error: Reported by OS = %d Expected = %d File = '%s''

##### Type: ERROR

##### Cause:

This event message is issued when a load memory from file command is processed and the size of the load file in bytes (as reported by the filesystem) doesn't match what would be expected based upon the load byte count specified in the file header.

The Reported by OS field holds size of the file in bytes as reported by the operating system, the Expected field holds the expected byte count that is the sum of number of load bytes specified in the file header and the size of the file header itself. The Name field holds the name of the load file that generated the error.

#### #define MM\_LD\_MEM\_FILE\_INF\_EID  5

'Load Memory From File Command: Loaded %d bytes to address 0x%08X from file '%s''

##### Event Message:

'Load Memory From File Command: Loaded %d bytes to address 0x%08X from file '%s''

##### Type: INFORMATIONAL

##### Cause:

This event message is issued when a load memory from file command has been executed.

The bytes field identifies how many bytes were written, the address field shows the fully resolved destination address of the load, the file field identifies the name of the file used for the load.

#### #define MM\_LEN\_ERR\_EID  18

'Invalid msg length: ID = 0x%04X, CC = %d, Len = %d, Expected = %d'

##### Event Message:

'Invalid msg length: ID = 0x%04X, CC = %d, Len = %d, Expected = %d'

##### Type: ERROR

##### Cause:

This event message is issued when command message is received with a message length that doesn't match the expected value.

The ID field contains the message ID, the CC field contains the command code, the Len field is the actual length returned by the CFE\_SB\_GetTotalMsgLength call, and the Expected field is the expected length for messages with that command code.

#### #define MM\_LOAD\_FILE\_CRC\_ERR\_EID  26

'Load file CRC failure: Expected = 0x%X Calculated = 0x%X File = '%s''

##### Event Message:

'Load file CRC failure: Expected = 0x%X Calculated = 0x%X File = '%s''

##### Type: ERROR

##### Cause:

This event message is issued when a CRC computation on the data in a load file does not return the expected result that is specified in the load file header.

The Expected field is the expected result, the Calculated field is the computed value, and File is the name of the file where the mismatch was detected

#### #define MM\_LOAD\_WID\_CRC\_ERR\_EID  27

'Interrupts Disabled Load CRC failure: Expected = 0x%X Calculated = 0x%X'

##### Event Message:

'Interrupts Disabled Load CRC failure: Expected = 0x%X Calculated = 0x%X'

##### Type: ERROR

##### Cause:

This event message is issued when a CRC computation on the data in a load with interrupts disabled command message does not return the expected result that is specified in the command message header.

The Expected field is the expected result and the Calculated field is the computed value.

#### #define MM\_LOAD\_WID\_INF\_EID  4

'Load Memory WID Command: Wrote %d bytes to address: 0x%08X'

##### Event Message:

'Load Memory WID Command: Wrote %d bytes to address: 0x%08X'

##### Type: INFORMATIONAL

##### Cause:

This event message is issued when a memory load with interrupts disabled command has been executed.

The bytes field identifies how many bytes were written, the address field shows the fully resolved destination address of the load.

#### #define MM\_MEMTYPE\_ERR\_EID  19

'Invalid memory type specified: MemType = %d'

##### Event Message:

'Invalid memory type specified: MemType = %d'

##### Type: ERROR

##### Cause:

This event message is issued when a command is received with an unrecognized or unsupported memory type specified.

The MemType field is the invalid memory type specifier that was received in the command message.

#### #define MM\_MID\_ERR\_EID  16

'Invalid command pipe message ID: 0x%X'

##### Event Message:

'Invalid command pipe message ID: 0x%X'

##### Type: ERROR

##### Cause:

This event message is issued when a software bus message is received with an invalid message ID.

The message ID field contains the message ID that generated the error.

#### #define MM\_NOOP\_INF\_EID  2

'No-op command. Version %d.%d.%d.%d'

##### Event Message:

'No-op command. Version %d.%d.%d.%d'

##### Type: INFORMATIONAL

##### Cause:

This event message is issued when a NOOP command has been received.

The Version fields contain the [MM\_MAJOR\_VERSION](#AAAAAAAAGS), [MM\_MINOR\_VERSION](#AAAAAAAAGT), [MM\_REVISION](#AAAAAAAAGU), and [MM\_MISSION\_REV](#AAAAAAAABR) version identifiers.

#### #define MM\_OS\_CLOSE\_ERR\_EID  33

'OS\_close error received: RC = 0x%08X File = '%s''

##### Event Message:

'OS\_close error received: RC = 0x%08X File = '%s''

##### Type: ERROR

##### Cause:

This event message is issued when a call to the #OS\_close function returns some value other than #OS\_SUCCESS.

The RC field is the return code and File is the filename from the #OS\_close call that generated the error.

#### #define MM\_OS\_CREAT\_ERR\_EID  31

'OS\_OpenCreate error received: RC = 0x%08X File = '%s''

##### Event Message:

'OS\_OpenCreate error received: RC = 0x%08X File = '%s''

##### Type: ERROR

##### Cause:

This event message is issued when a call to the #OS\_OpenCreate function returns some value other than #OS\_SUCCESS.

The RC field is the return code and File is the filename from the #OS\_OpenCreate call that generated the error.

#### #define MM\_OS\_EEPROMWRITE16\_ERR\_EID  29

'OS\_EepromWrite16 error received: RC = 0x%08X Addr = 0x%08X'

##### Event Message:

'OS\_EepromWrite16 error received: RC = 0x%08X Addr = 0x%08X'

##### Type: ERROR

##### Cause:

This event message is issued when a call to the #OS\_EepromWrite16 function returns some value other than #OS\_SUCCESS.

The RC field is the return code from the #OS\_EepromWrite16 call, the Addr field is the address that the write was attempted to.

#### #define MM\_OS\_EEPROMWRITE32\_ERR\_EID  30

'OS\_EepromWrite32 error received: RC = 0x%08X Addr = 0x%08X'

##### Event Message:

'OS\_EepromWrite32 error received: RC = 0x%08X Addr = 0x%08X'

##### Type: ERROR

##### Cause:

This event message is issued when a call to the #OS\_EepromWrite32 function returns some value other than #OS\_SUCCESS.

The RC field is the return code from the #OS\_EepromWrite32 call, the Addr field is the address that the write was attempted to.

#### #define MM\_OS\_EEPROMWRITE8\_ERR\_EID  28

'OS\_EepromWrite8 error received: RC = 0x%08X Addr = 0x%08X'

##### Event Message:

'OS\_EepromWrite8 error received: RC = 0x%08X Addr = 0x%08X'

##### Type: ERROR

##### Cause:

This event message is issued when a call to the #OS\_EepromWrite8 function returns some value other than #OS\_SUCCESS.

The RC field is the return code from the #OS\_EepromWrite8 call, the Addr field is the address that the write was attempted to.

#### #define MM\_OS\_MEMVALIDATE\_ERR\_EID  25

'CFE\_PSP\_MemValidate error received: RC = 0x%08X Addr = 0x%08X Size = %d MemType = %d'

##### Event Message:

'CFE\_PSP\_MemValidate error received: RC = 0x%08X Addr = 0x%08X Size = %d MemType = %d'

##### Type: ERROR

##### Cause:

This event message is issued when a call to the #CFE\_PSP\_MemValidateRange routine that is used to check address parameters fails.

The RC field is the return code from the #CFE\_PSP\_MemValidateRange call, the Addr field is the address, Size field is the specified size in bytes, and the MemType field is the memory type of the address range that failed validation.

#### #define MM\_OS\_OPEN\_ERR\_EID  32

'OS\_OpenCreate error received: RC = 0x%08X File = '%s''

##### Event Message:

'OS\_OpenCreate error received: RC = 0x%08X File = '%s''

##### Type: ERROR

##### Cause:

This event message is issued when a call to the #OS\_OpenCreate function returns some value other than #OS\_SUCCESS.

The RC field is the return code and File is the filename from the #OS\_OpenCreate call that generated the error.

#### #define MM\_OS\_READ\_ERR\_EID  34

'OS\_read error received: RC = 0x%08X File = '%s''

##### Event Message:

'OS\_read error received: RC = 0x%08X File = '%s''

##### Type: ERROR

##### Cause:

This event message is issued when a call to the #OS\_read function returns a negative error code.

The RC field is the return code and File is the filename from the #OS\_read call that generated the error.

#### #define MM\_OS\_READ\_EXP\_ERR\_EID  35

'OS\_read error received: RC = 0x%08X Expected = %d File = '%s''

##### Event Message:

'OS\_read error received: RC = 0x%08X Expected = %d File = '%s''

##### Type: ERROR

##### Cause:

This event message is issued when a call to the #OS\_read function returns some value other than the expected number of bytes read.

The RC field is the return code, the Expected field is the expected return value and the File is the filename from the #OS\_read call that generated the error.

#### #define MM\_OS\_STAT\_ERR\_EID  37

'OS\_stat error received: RC = 0x%08X File = '%s''

##### Event Message:

'OS\_stat error received: RC = 0x%08X File = '%s''

##### Type: ERROR

##### Cause:

This event message is issued when a call to the #OS\_stat function returns some value other than #OS\_SUCCESS.

The RC field is the return code and File is the filename from the #OS\_stat call that generated the error.

#### #define MM\_OS\_WRITE\_EXP\_ERR\_EID  36

'OS\_write error received: RC = 0x%08X Expected = %d File = '%s''

##### Event Message:

'OS\_write error received: RC = 0x%08X Expected = %d File = '%s''

##### Type: ERROR

##### Cause:

This event message is issued when a call to the #OS\_read function returns some value other than the expected number of bytes written.

The RC field is the return code, the Expected field is the expected return value and the File is the filename from the #OS\_write call that generated the error.

#### #define MM\_OS\_ZERO\_READ\_ERR\_EID  55

'Zero bytes read by OS\_read of file '%s''

##### Event Message:

'Zero bytes read by OS\_read of file '%s''

##### Type: ERROR

##### Cause:

This event message is issued when a call to the #OS\_read function returns zero total bytes read.

The File is the filename that #OS\_Read attempted to read.

#### #define MM\_PEEK\_BYTE\_INF\_EID  7

'Peek Command: Addr = 0x%08X Size = 8 bits Data = 0x%02X'

##### Event Message:

'Peek Command: Addr = 0x%08X Size = 8 bits Data = 0x%02X'

##### Type: INFORMATIONAL

##### Cause:

This event message is issued when an 8 bit memory peek command has been executed.

The Addr field shows the fully resolved address of the source memory location and the Data field contains the data read.

#### #define MM\_PEEK\_DWORD\_INF\_EID  9

'Peek Command: Addr = 0x%08X Size = 32 bits Data = 0x%08X'

##### Event Message:

'Peek Command: Addr = 0x%08X Size = 32 bits Data = 0x%08X'

##### Type: INFORMATIONAL

##### Cause:

This event message is issued when a 32 bit memory peek command has been executed.

The Addr field shows the fully resolved address of the source memory location and the Data field contains the data read.

#### #define MM\_PEEK\_WORD\_INF\_EID  8

'Peek Command: Addr = 0x%08X Size = 16 bits Data = 0x%04X'

##### Event Message:

'Peek Command: Addr = 0x%08X Size = 16 bits Data = 0x%04X'

##### Type: INFORMATIONAL

##### Cause:

This event message is issued when a 16 bit memory peek command has been executed.

The Addr field shows the fully resolved address of the source memory location and the Data field contains the data read.

#### #define MM\_PIPE\_ERR\_EID  15

'SB Pipe Read Error, App will exit. RC = 0x%08X'

##### Event Message:

'SB Pipe Read Error, App will exit. RC = 0x%08X'

##### Type: ERROR

##### Cause:

This event message is issued when a call to #CFE\_SB\_RcvMsg fails.

The RC field is the return code from the #CFE\_SB\_RcvMsg function call that generated the error.

#### #define MM\_POKE\_BYTE\_INF\_EID  10

'Poke Command: Addr = 0x%08X Size = 8 bits Data = 0x%02X'

##### Event Message:

'Poke Command: Addr = 0x%08X Size = 8 bits Data = 0x%02X'

##### Type: INFORMATIONAL

##### Cause:

This event message is issued when an 8 bit memory poke command has been executed.

The Addr field shows the fully resolved address of the destination memory location and the Data field contains the data written.

#### #define MM\_POKE\_DWORD\_INF\_EID  12

'Poke Command: Addr = 0x%08X Size = 32 bits Data = 0x%08X'

##### Event Message:

'Poke Command: Addr = 0x%08X Size = 32 bits Data = 0x%08X'

##### Type: INFORMATIONAL

##### Cause:

This event message is issued when a 32 bit memory poke command has been executed.

The Addr field shows the fully resolved address of the destination memory location and the Data field contains the data written.

#### #define MM\_POKE\_WORD\_INF\_EID  11

'Poke Command: Addr = 0x%08X Size = 16 bits Data = 0x%04X'

##### Event Message:

'Poke Command: Addr = 0x%08X Size = 16 bits Data = 0x%04X'

##### Type: INFORMATIONAL

##### Cause:

This event message is issued when a 16 bit memory poke command has been executed.

The Addr field shows the fully resolved address of the destination memory location and the Data field contains the data written.

#### #define MM\_PSP\_READ\_ERR\_EID  56

'PSP read memory error: RC=0x%08X, Src=0x%08X, Tgt=0x%08X, Type='s''

##### Event Message:

'PSP read memory error: RC=0x%08X, Src=0x%08X, Tgt=0x%08X, Type='s''

##### Type: ERROR

##### Cause:

This event message is issued when a call to one of the CFE\_PSP\_MemRead functions (#CFE\_PSP\_MemRead8, #CFE\_PSP\_MemRead16, #CFE\_PSP\_MemRead32) returns something other than CFE\_PSP\_SUCCESS.

The RC field is the function return code and Src is the read location and Tgt is the storage location from the function call that generated the error. The Type field will indicate MEM8, MEM16 or MEM32.

#### #define MM\_PSP\_WRITE\_ERR\_EID  57

'PSP write memory error: RC=0x%08X, Address=0x%08X, MemType='s''

##### Event Message:

'PSP write memory error: RC=0x%08X, Address=0x%08X, MemType='s''

##### Type: ERROR

##### Cause:

This event message is issued when a call to one of the CFE\_PSP\_MemWrite functions (#CFE\_PSP\_MemWrite8, #CFE\_PSP\_MemWrite16, #CFE\_PSP\_MemWrite32) returns something other than CFE\_PSP\_SUCCESS.

The RC field is the function return code and Address is the write location from the function call that generated the error. The MemType field will indicate MEM8, MEM16 or MEM32.

#### #define MM\_RESET\_INF\_EID  3

'Reset counters command received'

##### Event Message:

'Reset counters command received'

##### Type: INFORMATIONAL

##### Cause:

This event message is issued when a reset counters command has been received.

#### #define MM\_SYM\_LOOKUP\_INF\_EID  45

'Symbol Lookup Command: Name = '%s' Addr = 0x%08X'

##### Event Message:

'Symbol Lookup Command: Name = '%s' Addr = 0x%08X'

##### Type: INFORMATIONAL

##### Cause:

This event message is issued when a symbol lookup command has been successfully executed.

The Name field holds the symbol name string, the Addr field shows the fully resolved address

#### #define MM\_SYMFILENAME\_NUL\_ERR\_EID  48

'NUL (empty) string specified as symbol dump file name'

##### Event Message:

'NUL (empty) string specified as symbol dump file name'

##### Type: ERROR

##### Cause:

This event message is issued when a NUL string has been specified as the dump file name in a dump symbol table to file command

#### #define MM\_SYMNAME\_ERR\_EID  20

'Symbolic address can't be resolved: Name = '%s''

##### Event Message:

'Symbolic address can't be resolved: Name = '%s''

##### Type: ERROR

##### Cause:

This event message is issued when a symbol name string can't be resolved by the OSAPI.

The Name field is the symbol name string that generated the error.

#### #define MM\_SYMNAME\_NUL\_ERR\_EID  46

'NUL (empty) string specified as symbol name'

##### Event Message:

'NUL (empty) string specified as symbol name'

##### Type: ERROR

##### Cause:

This event message is issued when a NUL string has been specified as the symbol name in a lookup symbol command

#### #define MM\_SYMTBL\_TO\_FILE\_FAIL\_ERR\_EID  49

'Error dumping symbol table, OS\_Status= 0xX, File='s''

##### Event Message:

'Error dumping symbol table, OS Status= 0xX, File='s''

##### Type: ERROR

##### Cause:

This event message is issued when a NUL string has been specified as the dump file name in a dump symbol table to file command

The OS\_Status field holds the return code from the call to #OS\_SymbolTableDump The File field holds the requested dump file name string

#### #define MM\_SYMTBL\_TO\_FILE\_INF\_EID  47

'Symbol Table Dump to File Started: Name = '%s''

##### Event Message:

'Symbol Table Dump to File Started: Name = '%s''

##### Type: INFORMATIONAL

##### Cause:

This event message is issued when a dump symbol table fo file command has been successfully executed.

The Name field holds the requested dump file name string

#### #define MM\_SYMTBL\_TO\_FILE\_INVALID\_ERR\_EID  50

'Illegal characters in target filename, File='s''

##### Event Message:

'Illegal characters in target filename, File='s''

##### Type: ERROR

##### Cause:

This event message is issued when an illegal character has been found in the specified dump file name of a dump symbol table to file command

The File field holds the requested dump file name string

## fsw/src/mm\_filedefs.h File Reference

#include "mm\_msg.h"

### Data Structures

* struct [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM)

*MM Load and Dump file header structure We use the same header structure for both dump and load files so a dump file can be reloaded back into memory if desired (providing a memory save and restore capability). This MM header is the secondary header, the standard cFE file header is primary header for all load and dump files.*

## fsw/src/mm\_load.c File Reference

#include "mm\_app.h"

#include "mm\_load.h"

#include "mm\_perfids.h"

#include "mm\_events.h"

#include "mm\_utils.h"

#include "mm\_mem32.h"

#include "mm\_mem16.h"

#include "mm\_mem8.h"

#include "mm\_mission\_cfg.h"

#include "cfs\_utils.h"

#include <string.h>

### Functions

* bool [MM\_PokeCmd](#AAAAAAAAGX) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process memory poke command.*

* bool [MM\_PokeMem](#AAAAAAAAGY) (const [MM\_PokeCmd\_t](#AAAAAAAAGZ) \*CmdPtr, cpuaddr DestAddress)

*Memory poke.*

* bool [MM\_PokeEeprom](#AAAAAAAAHA) (const [MM\_PokeCmd\_t](#AAAAAAAAGZ) \*CmdPtr, cpuaddr DestAddress)

*Eeprom poke.*

* bool [MM\_LoadMemWIDCmd](#AAAAAAAAHB) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process load memory with interrupts disabled command.*

* bool [MM\_LoadMemFromFileCmd](#AAAAAAAAHC) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process memory load from file command.*

* bool [MM\_LoadMemFromFile](#AAAAAAAAHD) (uint32 FileHandle, const char \*FileName, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \*FileHeader, cpuaddr DestAddress)

*Memory load from file.*

* bool [MM\_VerifyLoadFileSize](#AAAAAAAAHE) (const char \*FileName, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \*FileHeader)

*Verify load file size.*

* bool [MM\_ReadFileHeaders](#AAAAAAAAHF) (const char \*FileName, int32 FileHandle, CFE\_FS\_Header\_t \*CFEHeader, [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \*MMHeader)

*Read the cFE primary and and MM secondary file headers.*

* bool [MM\_FillMemCmd](#AAAAAAAAHG) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process memory fill command.*

* bool [MM\_FillMem](#AAAAAAAAHH) (cpuaddr DestAddress, const [MM\_FillMemCmd\_t](#AAAAAAAAHI) \*CmdPtr)

*Fill memory.*

### Variables

* [MM\_AppData\_t](#AAAAAAAACD) [MM\_AppData](#AAAAAAAAHJ)

### Function Documentation

#### bool MM\_FillMem (cpuaddr *DestAddr*, const [MM\_FillMemCmd\_t](#AAAAAAAAHI) \* *CmdPtr*)

Fill memory.

##### Description

Support function for [MM\_FillMemCmd](#AAAAAAAAHK). This routine will load memory with a command specified fill pattern

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *DestAddr* | The destination address for the fill operation |
| in | *CmdPtr* | A [MM\_FillMemCmd\_t](#AAAAAAAAHI) pointer to the fill command message |

#### bool MM\_FillMemCmd (const CFE\_SB\_Buffer\_t \* *BufPtr*)

Process memory fill command.

##### Description

Processes the memory fill command that will load an address range of memory with the command specified fill pattern

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *BufPtr* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_FILL\_MEM\_CC](#AAAAAAAAHL)

#### bool MM\_LoadMemFromFile (uint32 *FileHandle*, const char \* *FileName*, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \* *FileHeader*, cpuaddr *DestAddress*)

Memory load from file.

##### Description

Support function for [MM\_LoadMemFromFileCmd](#AAAAAAAAHM). This routine will read a file and write the data to memory

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *FileHandle* | The open file handle of the load file |
| in | *FileName* | A pointer to a character string holding the load file name |
| in | *FileHeader* | A [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) pointer to the load file header structure |
| in | *DestAddress* | The destination address for the requested load operation |

##### Returns:

Returns TRUE if the load completed successfully Returns FALSE if the load failed due to an error

#### bool MM\_LoadMemFromFileCmd (const CFE\_SB\_Buffer\_t \* *BufPtr*)

Process memory load from file command.

##### Description

Processes the memory load from file command that will read a file and store the data in the command specified address range of memory.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *BufPtr* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_LOAD\_MEM\_FROM\_FILE\_CC](#AAAAAAAAHN)

#### bool MM\_LoadMemWIDCmd (const CFE\_SB\_Buffer\_t \* *BufPtr*)

Process load memory with interrupts disabled command.

##### Description

Processes the load memory with interrupts disabled command that will load up to [MM\_MAX\_UNINTERRUPTIBLE\_DATA](#AAAAAAAAAV) bytes into ram with interrupts disabled during the actual load

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *BufPtr* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_LOAD\_MEM\_WID\_CC](#AAAAAAAAAF)

#### bool MM\_PokeCmd (const CFE\_SB\_Buffer\_t \* *BufPtr*)

Process memory poke command.

##### Description

Processes the memory poke command that will load a memory location with data specified in the command message.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *BufPtr* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_POKE\_CC](#AAAAAAAAHO)

#### bool MM\_PokeEeprom (const [MM\_PokeCmd\_t](#AAAAAAAAGZ) \* *CmdPtr*, cpuaddr *DestAddress*)

Eeprom poke.

##### Description

Support function for [MM\_PokeCmd](#AAAAAAAAHP). This routine will write 8, 16, or 32 bits of data to a single EEPROM address.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *CmdPtr* | A [MM\_PokeCmd\_t](#AAAAAAAAGZ) pointer to the poke command message |
| in | *DestAddress* | The destination address for the poke operation |

#### bool MM\_PokeMem (const [MM\_PokeCmd\_t](#AAAAAAAAGZ) \* *CmdPtr*, cpuaddr *DestAddress*)

Memory poke.

##### Description

Support function for [MM\_PokeCmd](#AAAAAAAAHP). This routine will write 8, 16, or 32 bits of data to a single ram address.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *CmdPtr* | A [MM\_PokeCmd\_t](#AAAAAAAAGZ) pointer to the poke command message |
| in | *DestAddress* | The destination address for the poke operation |

#### bool MM\_ReadFileHeaders (const char \* *FileName*, int32 *FileHandle*, CFE\_FS\_Header\_t \* *CFEHeader*, [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \* *MMHeader*)

Read the cFE primary and and MM secondary file headers.

##### Description

Support function for [MM\_LoadMemFromFileCmd](#AAAAAAAAHM). This routine will read the cFE primary and MM secondary headers from the file specified by the FileHandle.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *FileName* | A pointer to a character string holding the file name (used only for error event messages). |
| in | *FileHandle* | File Descriptor obtained from a previous call to #OS\_open that is associated with the file whose headers are to be read. |
| in | *CFEHeader* | A #CFE\_FS\_Header\_t pointer to the cFE primary file header structure. |
| in | *MMHeader* | A [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) pointer to the MM secondary file header structure |
| out | *\*CFEHeader* | Contents of the cFE primary file header structure for the specified file. |
| out | *\*MMHeader* | Contents of the MM secondary file header structure for the specified file. |

##### Returns:

Returns TRUE if the headers were read successfully Returns FALSE if a read error occurred

#### bool MM\_VerifyLoadFileSize (const char \* *FileName*, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \* *FileHeader*)

Verify load file size.

##### Description

Support function for [MM\_LoadMemFromFileCmd](#AAAAAAAAHM). This routine will check if the size of a load file as reported by the filesystem is what it should be based upon the number of load bytes specified in the MM file header.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *FileName* | A pointer to a character string holding the load file name |
| in | *FileHeader* | A [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) pointer to the load file header structure |

##### Returns:

Returns TRUE if the load file size is as expected Returns FALSE if the load file size is not as expected

### Variable Documentation

#### [MM\_AppData\_t](#AAAAAAAACD) MM\_AppData

#### 

## fsw/src/mm\_load.h File Reference

#include "cfe.h"

#include "mm\_msg.h"

#include "mm\_filedefs.h"

### Functions

* bool [MM\_PokeMem](#AAAAAAAAHR) (const [MM\_PokeCmd\_t](#AAAAAAAAGZ) \*CmdPtr, cpuaddr DestAddress)

*Memory poke.*

* bool [MM\_PokeEeprom](#AAAAAAAAHS) (const [MM\_PokeCmd\_t](#AAAAAAAAGZ) \*CmdPtr, cpuaddr DestAddress)

*Eeprom poke.*

* bool [MM\_LoadMemWID](#AAAAAAAAHT) (const [MM\_LoadMemWIDCmd\_t](#AAAAAAAAHU) \*CmdPtr, cpuaddr DestAddress)

*Load memory with interrupts disabled.*

* bool [MM\_LoadMemFromFile](#AAAAAAAAHV) (uint32 FileHandle, const char \*FileName, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \*FileHeader, cpuaddr DestAddress)

*Memory load from file.*

* bool [MM\_VerifyLoadFileSize](#AAAAAAAAHW) (const char \*FileName, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \*FileHeader)

*Verify load file size.*

* bool [MM\_ReadFileHeaders](#AAAAAAAAHX) (const char \*FileName, int32 FileHandle, CFE\_FS\_Header\_t \*CFEHeader, [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \*MMHeader)

*Read the cFE primary and and MM secondary file headers.*

* bool [MM\_FillMem](#AAAAAAAAHY) (cpuaddr DestAddr, const [MM\_FillMemCmd\_t](#AAAAAAAAHI) \*CmdPtr)

*Fill memory.*

* bool [MM\_PokeCmd](#AAAAAAAAHP) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process memory poke command.*

* bool [MM\_LoadMemWIDCmd](#AAAAAAAAHZ) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process load memory with interrupts disabled command.*

* bool [MM\_LoadMemFromFileCmd](#AAAAAAAAHM) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process memory load from file command.*

* bool [MM\_FillMemCmd](#AAAAAAAAHK) (const CFE\_SB\_Buffer\_t \*BufPtr)

*Process memory fill command.*

### Function Documentation

#### bool MM\_FillMem (cpuaddr *DestAddr*, const [MM\_FillMemCmd\_t](#AAAAAAAAHI) \* *CmdPtr*)

Fill memory.

##### Description

Support function for [MM\_FillMemCmd](#AAAAAAAAHK). This routine will load memory with a command specified fill pattern

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *DestAddr* | The destination address for the fill operation |
| in | *CmdPtr* | A [MM\_FillMemCmd\_t](#AAAAAAAAHI) pointer to the fill command message |

#### bool MM\_FillMemCmd (const CFE\_SB\_Buffer\_t \* *BufPtr*)

Process memory fill command.

##### Description

Processes the memory fill command that will load an address range of memory with the command specified fill pattern

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *BufPtr* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_FILL\_MEM\_CC](#AAAAAAAAHL)

#### bool MM\_LoadMemFromFile (uint32 *FileHandle*, const char \* *FileName*, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \* *FileHeader*, cpuaddr *DestAddress*)

Memory load from file.

##### Description

Support function for [MM\_LoadMemFromFileCmd](#AAAAAAAAHM). This routine will read a file and write the data to memory

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *FileHandle* | The open file handle of the load file |
| in | *FileName* | A pointer to a character string holding the load file name |
| in | *FileHeader* | A [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) pointer to the load file header structure |
| in | *DestAddress* | The destination address for the requested load operation |

##### Returns:

Returns TRUE if the load completed successfully Returns FALSE if the load failed due to an error

#### bool MM\_LoadMemFromFileCmd (const CFE\_SB\_Buffer\_t \* *BufPtr*)

Process memory load from file command.

##### Description

Processes the memory load from file command that will read a file and store the data in the command specified address range of memory.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *BufPtr* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_LOAD\_MEM\_FROM\_FILE\_CC](#AAAAAAAAHN)

#### bool MM\_LoadMemWID (const [MM\_LoadMemWIDCmd\_t](#AAAAAAAAHU) \* *CmdPtr*, cpuaddr *DestAddress*)

Load memory with interrupts disabled.

##### Description

Support function for [MM\_LoadMemWIDCmd](#AAAAAAAAHZ). This routine will load up to [MM\_MAX\_UNINTERRUPTIBLE\_DATA](#AAAAAAAAAV) bytes into ram with interrupts disabled during the actual load

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *CmdPtr* | A [MM\_LoadMemWIDCmd\_t](#AAAAAAAAHU) pointer to the command message |
| in | *DestAddress* | The destination address for the load operation |

#### bool MM\_LoadMemWIDCmd (const CFE\_SB\_Buffer\_t \* *BufPtr*)

Process load memory with interrupts disabled command.

##### Description

Processes the load memory with interrupts disabled command that will load up to [MM\_MAX\_UNINTERRUPTIBLE\_DATA](#AAAAAAAAAV) bytes into ram with interrupts disabled during the actual load

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *BufPtr* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_LOAD\_MEM\_WID\_CC](#AAAAAAAAAF)

#### bool MM\_PokeCmd (const CFE\_SB\_Buffer\_t \* *BufPtr*)

Process memory poke command.

##### Description

Processes the memory poke command that will load a memory location with data specified in the command message.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *BufPtr* | A #CFE\_SB\_Buffer\_t\* pointer that references the software bus message |

##### See also:

[MM\_POKE\_CC](#AAAAAAAAHO)

#### bool MM\_PokeEeprom (const [MM\_PokeCmd\_t](#AAAAAAAAGZ) \* *CmdPtr*, cpuaddr *DestAddress*)

Eeprom poke.

##### Description

Support function for [MM\_PokeCmd](#AAAAAAAAHP). This routine will write 8, 16, or 32 bits of data to a single EEPROM address.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *CmdPtr* | A [MM\_PokeCmd\_t](#AAAAAAAAGZ) pointer to the poke command message |
| in | *DestAddress* | The destination address for the poke operation |

#### bool MM\_PokeMem (const [MM\_PokeCmd\_t](#AAAAAAAAGZ) \* *CmdPtr*, cpuaddr *DestAddress*)

Memory poke.

##### Description

Support function for [MM\_PokeCmd](#AAAAAAAAHP). This routine will write 8, 16, or 32 bits of data to a single ram address.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *CmdPtr* | A [MM\_PokeCmd\_t](#AAAAAAAAGZ) pointer to the poke command message |
| in | *DestAddress* | The destination address for the poke operation |

#### bool MM\_ReadFileHeaders (const char \* *FileName*, int32 *FileHandle*, CFE\_FS\_Header\_t \* *CFEHeader*, [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \* *MMHeader*)

Read the cFE primary and and MM secondary file headers.

##### Description

Support function for [MM\_LoadMemFromFileCmd](#AAAAAAAAHM). This routine will read the cFE primary and MM secondary headers from the file specified by the FileHandle.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *FileName* | A pointer to a character string holding the file name (used only for error event messages). |
| in | *FileHandle* | File Descriptor obtained from a previous call to #OS\_open that is associated with the file whose headers are to be read. |
| in | *CFEHeader* | A #CFE\_FS\_Header\_t pointer to the cFE primary file header structure. |
| in | *MMHeader* | A [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) pointer to the MM secondary file header structure |
| out | *\*CFEHeader* | Contents of the cFE primary file header structure for the specified file. |
| out | *\*MMHeader* | Contents of the MM secondary file header structure for the specified file. |

##### Returns:

Returns TRUE if the headers were read successfully Returns FALSE if a read error occurred

#### bool MM\_VerifyLoadFileSize (const char \* *FileName*, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \* *FileHeader*)

Verify load file size.

##### Description

Support function for [MM\_LoadMemFromFileCmd](#AAAAAAAAHM). This routine will check if the size of a load file as reported by the filesystem is what it should be based upon the number of load bytes specified in the MM file header.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *FileName* | A pointer to a character string holding the load file name |
| in | *FileHeader* | A [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) pointer to the load file header structure |

##### Returns:

Returns TRUE if the load file size is as expected Returns FALSE if the load file size is not as expected

## fsw/src/mm\_mem16.c File Reference

#include "mm\_mem16.h"

#include "mm\_app.h"

#include "mm\_events.h"

#include "mm\_utils.h"

#include <string.h>

## fsw/src/mm\_mem16.h File Reference

#include "mm\_filedefs.h"

### Functions

* bool [MM\_LoadMem16FromFile](#AAAAAAAAIC) (uint32 FileHandle, const char \*FileName, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \*FileHeader, cpuaddr DestAddress)

*Memory16 load from file.*

* bool [MM\_DumpMem16ToFile](#AAAAAAAAID) (uint32 FileHandle, const char \*FileName, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \*FileHeader)

*Memory16 dump to file.*

* bool [MM\_FillMem16](#AAAAAAAAGQ) (cpuaddr DestAddress, const [MM\_FillMemCmd\_t](#AAAAAAAAHI) \*CmdPtr)

*Fill memory16.*

### Function Documentation

#### bool MM\_DumpMem16ToFile (uint32 *FileHandle*, const char \* *FileName*, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \* *FileHeader*)

Memory16 dump to file.

##### Description

Support function for [MM\_DumpMemToFileCmd](#AAAAAAAADU). This routine will read an address range that is defined to only be 16 bit accessible and store the data in a file

##### Assumptions, External Events, and Notes:

This function is specific to the optional [MM\_MEM16](#AAAAAAAAIE) memory type

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *FileHandle* | The open file handle of the dump file |
| in | *FileName* | A pointer to a character string holding the dump file name |
| in | *FileHeader* | A [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) pointer to the dump file header structure initialized with data based upon the command message parameters |

##### Returns:

Returns TRUE if the dump completed successfully Returns FALSE if the dump failed due to an error

#### bool MM\_FillMem16 (cpuaddr *DestAddress*, const [MM\_FillMemCmd\_t](#AAAAAAAAHI) \* *CmdPtr*)

Fill memory16.

##### Description

Support function for [MM\_FillMemCmd](#AAAAAAAAHK). This routine will load memory that is defined to only be 16 bit accessible with a command specified fill pattern

##### Assumptions, External Events, and Notes:

This function is specific to the optional [MM\_MEM16](#AAAAAAAAIE) memory type

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *DestAddress* | The destination address for the fill operation |
| in | *CmdPtr* | A [MM\_FillMemCmd\_t](#AAAAAAAAHI) pointer to the fill command message |

#### bool MM\_LoadMem16FromFile (uint32 *FileHandle*, const char \* *FileName*, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \* *FileHeader*, cpuaddr *DestAddress*)

Memory16 load from file.

##### Description

Support function for [MM\_LoadMemFromFileCmd](#AAAAAAAAHM). This routine will read a file and write the data to memory that is defined to only be 16 bit accessible

##### Assumptions, External Events, and Notes:

This function is specific to the optional [MM\_MEM16](#AAAAAAAAIE) memory type

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *FileHandle* | The open file handle of the load file |
| in | *FileName* | A pointer to a character string holding the load file name |
| in | *FileHeader* | A [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) pointer to the load file header structure |
| in | *DestAddress* | The destination address for the requested load operation |

##### Returns:

Returns TRUE if the load completed successfully Returns FALSE if the load failed due to an error

## fsw/src/mm\_mem32.c File Reference

#include "mm\_mem32.h"

#include "mm\_app.h"

#include "mm\_events.h"

#include "mm\_utils.h"

#include <string.h>

## fsw/src/mm\_mem32.h File Reference

#include "mm\_filedefs.h"

### Functions

* bool [MM\_LoadMem32FromFile](#AAAAAAAAIH) (uint32 FileHandle, const char \*FileName, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \*FileHeader, cpuaddr DestAddress)

*Memory32 load from file.*

* bool [MM\_DumpMem32ToFile](#AAAAAAAAII) (uint32 FileHandle, const char \*FileName, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \*FileHeader)

*Memory32 dump to file.*

* bool [MM\_FillMem32](#AAAAAAAAGR) (cpuaddr DestAddress, const [MM\_FillMemCmd\_t](#AAAAAAAAHI) \*CmdPtr)

*Fill memory32.*

### Function Documentation

#### bool MM\_DumpMem32ToFile (uint32 *FileHandle*, const char \* *FileName*, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \* *FileHeader*)

Memory32 dump to file.

##### Description

Support function for [MM\_DumpMemToFileCmd](#AAAAAAAADU). This routine will read an address range that is defined to only be 32 bit accessible and store the data in a file

##### Assumptions, External Events, and Notes:

This function is specific to the optional [MM\_MEM32](#AAAAAAAAIJ) memory type

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *FileHandle* | The open file handle of the dump file |
| in | *FileName* | A pointer to a character string holding the dump file name |
| in | *FileHeader* | A [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) pointer to the dump file header structure initialized with data based upon the command message parameters |

##### Returns:

Returns TRUE if the dump completed successfully Returns FALSE if the dump failed due to an error

#### bool MM\_FillMem32 (cpuaddr *DestAddress*, const [MM\_FillMemCmd\_t](#AAAAAAAAHI) \* *CmdPtr*)

Fill memory32.

##### Description

Support function for [MM\_FillMemCmd](#AAAAAAAAHK). This routine will load memory that is defined to only be 32 bit accessible with a command specified fill pattern

##### Assumptions, External Events, and Notes:

This function is specific to the optional [MM\_MEM32](#AAAAAAAAIJ) memory type

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *DestAddress* | The destination address for the fill operation |
| in | *CmdPtr* | A [MM\_FillMemCmd\_t](#AAAAAAAAHI) pointer to the fill command message |

#### bool MM\_LoadMem32FromFile (uint32 *FileHandle*, const char \* *FileName*, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \* *FileHeader*, cpuaddr *DestAddress*)

Memory32 load from file.

##### Description

Support function for [MM\_LoadMemFromFileCmd](#AAAAAAAAHM). This routine will read a file and write the data to memory that is defined to only be 32 bit accessible

##### Assumptions, External Events, and Notes:

This function is specific to the optional [MM\_MEM32](#AAAAAAAAIJ) memory type

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *FileHandle* | The open file handle of the load file |
| in | *FileName* | A pointer to a character string holding the load file name |
| in | *FileHeader* | A [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) pointer to the load file header structure |
| in | *DestAddress* | The destination address for the requested load operation |

##### Returns:

Returns TRUE if the load completed successfully Returns FALSE if the load failed due to an error

## fsw/src/mm\_mem8.c File Reference

#include "mm\_mem8.h"

#include "mm\_app.h"

#include "mm\_events.h"

#include "mm\_utils.h"

#include <string.h>

## fsw/src/mm\_mem8.h File Reference

#include "mm\_filedefs.h"

### Functions

* bool [MM\_LoadMem8FromFile](#AAAAAAAAIM) (uint32 FileHandle, const char \*FileName, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \*FileHeader, cpuaddr DestAddress)

*Memory8 load from file.*

* bool [MM\_DumpMem8ToFile](#AAAAAAAAIN) (uint32 FileHandle, const char \*FileName, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \*FileHeader)

*Memory8 dump to file.*

* bool [MM\_FillMem8](#AAAAAAAAIO) (cpuaddr DestAddress, const [MM\_FillMemCmd\_t](#AAAAAAAAHI) \*CmdPtr)

*Fill memory8.*

### Function Documentation

#### bool MM\_DumpMem8ToFile (uint32 *FileHandle*, const char \* *FileName*, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \* *FileHeader*)

Memory8 dump to file.

##### Description

Support function for [MM\_DumpMemToFileCmd](#AAAAAAAADU). This routine will read an address range that is defined to only be 8 bit accessible and store the data in a file

##### Assumptions, External Events, and Notes:

This function is specific to the optional [MM\_MEM8](#AAAAAAAAIP) memory type

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *FileHandle* | The open file handle of the dump file |
| in | *FileName* | A pointer to a character string holding the dump file name |
| in | *FileHeader* | A [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) pointer to the dump file header structure initialized with data based upon the command message parameters |

##### Returns:

Returns TRUE if the dump completed successfully Returns FALSE if the dump failed due to an error

#### bool MM\_FillMem8 (cpuaddr *DestAddress*, const [MM\_FillMemCmd\_t](#AAAAAAAAHI) \* *CmdPtr*)

Fill memory8.

##### Description

Support function for [MM\_FillMemCmd](#AAAAAAAAHK). This routine will load memory that is defined to only be 8 bit accessible with a command specified fill pattern

##### Assumptions, External Events, and Notes:

This function is specific to the optional [MM\_MEM8](#AAAAAAAAIP) memory type

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *DestAddress* | The destination address for the fill operation |
| in | *CmdPtr* | A [MM\_FillMemCmd\_t](#AAAAAAAAHI) pointer to the fill command message |

#### bool MM\_LoadMem8FromFile (uint32 *FileHandle*, const char \* *FileName*, const [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) \* *FileHeader*, cpuaddr *DestAddress*)

Memory8 load from file.

##### Description

Support function for [MM\_LoadMemFromFileCmd](#AAAAAAAAHM). This routine will read a file and write the data to memory that is defined to only be 8 bit accessible

##### Assumptions, External Events, and Notes:

This function is specific to the optional [MM\_MEM8](#AAAAAAAAIP) memory type

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *FileHandle* | The open file handle of the load file |
| in | *FileName* | A pointer to a character string holding the load file name |
| in | *FileHeader* | A [MM\_LoadDumpFileHeader\_t](#AAAAAAAADM) pointer to the load file header structure |
| in | *DestAddress* | The destination address for the requested load operation |

##### Returns:

Returns TRUE if the load completed successfully Returns FALSE if the load failed due to an error

## fsw/src/mm\_msg.h File Reference

#include "mm\_platform\_cfg.h"

#include "cfs\_utils.h"

#include "cfe.h"

#include "mm\_msgdefs.h"

### Data Structures

* struct [MM\_NoArgsCmd\_t](#AAAAAAAAIR)
* No Arguments Command For command details see [MM\_NOOP\_CC](#AAAAAAAACI), [MM\_RESET\_CC](#AAAAAAAACJ). struct [MM\_PeekCmd\_t](#AAAAAAAADJ)
* Memory Peek Command For command details see [MM\_PEEK\_CC](#AAAAAAAADX). struct [MM\_PokeCmd\_t](#AAAAAAAAGZ)
* Memory Poke Command For command details see [MM\_POKE\_CC](#AAAAAAAAHO). struct [MM\_LoadMemWIDCmd\_t](#AAAAAAAAHU)
* Memory Load With Interrupts Disabled Command For command details see [MM\_LOAD\_MEM\_WID\_CC](#AAAAAAAAAF). struct [MM\_DumpInEventCmd\_t](#AAAAAAAADQ)
* Dump Memory In Event Message Command For command details see [MM\_DUMP\_IN\_EVENT\_CC](#AAAAAAAADT). struct [MM\_LoadMemFromFileCmd\_t](#AAAAAAAAIS)
* Memory Load From File Command For command details see [MM\_LOAD\_MEM\_FROM\_FILE\_CC](#AAAAAAAAHN). struct [MM\_DumpMemToFileCmd\_t](#AAAAAAAAIT)
* Memory Dump To File Command For command details see [MM\_DUMP\_MEM\_TO\_FILE\_CC](#AAAAAAAADV). struct [MM\_FillMemCmd\_t](#AAAAAAAAHI)
* Memory Fill Command For command details see [MM\_FILL\_MEM\_CC](#AAAAAAAAHL). struct [MM\_LookupSymCmd\_t](#AAAAAAAAIU)
* Symbol Table Lookup Command For command details see [MM\_LOOKUP\_SYM\_CC](#AAAAAAAACH). struct [MM\_SymTblToFileCmd\_t](#AAAAAAAAIV)
* Save Symbol Table To File Command For command details see [MM\_SYMTBL\_TO\_FILE\_CC](#AAAAAAAACK). struct [MM\_EepromWriteEnaCmd\_t](#AAAAAAAAIW)
* EEPROM Write Enable Command For command details see [MM\_ENABLE\_EEPROM\_WRITE\_CC](#AAAAAAAACG). struct [MM\_EepromWriteDisCmd\_t](#AAAAAAAAIX)
* EEPROM Write Disable Command For command details see [MM\_DISABLE\_EEPROM\_WRITE\_CC](#AAAAAAAACF). struct [MM\_HkPacket\_t](#AAAAAAAAIY)

## fsw/src/mm\_msgdefs.h File Reference

### Macros

* #define [MM\_NOOP\_CC](#AAAAAAAACI)  0
* #define [MM\_RESET\_CC](#AAAAAAAACJ)  1
* #define [MM\_PEEK\_CC](#AAAAAAAADX)  2
* #define [MM\_POKE\_CC](#AAAAAAAAHO)  3
* #define [MM\_LOAD\_MEM\_WID\_CC](#AAAAAAAAAF)  4
* #define [MM\_LOAD\_MEM\_FROM\_FILE\_CC](#AAAAAAAAHN)  5
* #define [MM\_DUMP\_MEM\_TO\_FILE\_CC](#AAAAAAAADV)  6
* #define [MM\_DUMP\_IN\_EVENT\_CC](#AAAAAAAADT)  7
* #define [MM\_FILL\_MEM\_CC](#AAAAAAAAHL)  8
* #define [MM\_LOOKUP\_SYM\_CC](#AAAAAAAACH)  9
* #define [MM\_SYMTBL\_TO\_FILE\_CC](#AAAAAAAACK)  10
* #define [MM\_ENABLE\_EEPROM\_WRITE\_CC](#AAAAAAAACG)  11
* #define [MM\_DISABLE\_EEPROM\_WRITE\_CC](#AAAAAAAACF)  12

MM Data Sizes for Peeks and Pokes#define [MM\_BYTE\_BIT\_WIDTH](#AAAAAAAAJA)  8

#define [MM\_WORD\_BIT\_WIDTH](#AAAAAAAAJB)  16

#define [MM\_DWORD\_BIT\_WIDTH](#AAAAAAAAJC)  32

MM Memory Types#define [MM\_NOMEMTYPE](#AAAAAAAAJD)  0

*Used to indicate that no memtype specified.*

#define [MM\_RAM](#AAAAAAAAJE)  1

*Normal RAM, no special access required.*

#define [MM\_EEPROM](#AAAAAAAACN)  2

*EEPROM, requires special access for writes.*

#define [MM\_MEM8](#AAAAAAAAIP)  3

*Optional memory type that is only 8-bit read/write.*

#define [MM\_MEM16](#AAAAAAAAIE)  4

*Optional memory type that is only 16-bit read/write.*

#define [MM\_MEM32](#AAAAAAAAIJ)  5

*Optional memory type that is only 32-bit read/write.*

#define [MM\_NUM\_MEMTYPES](#AAAAAAAAJF)  6

Misc Initialization Values#define [MM\_CLEAR\_SYMNAME](#AAAAAAAAJG)  '\0'

*Used to clear out symbol name strings.*

#define [MM\_CLEAR\_FNAME](#AAAAAAAAJH)  '\0'

*Used to clear out file name strings.*

#define [MM\_CLEAR\_ADDR](#AAAAAAAAJI)  0

*Used to clear out memory address variables.*

#define [MM\_CLEAR\_PATTERN](#AAAAAAAAJJ)  0

*Used to clear out fill and test patterns.*

HK MM Last Action Identifiers#define [MM\_NOACTION](#AAAAAAAAJK)  0

*Used to clear out HK action variable.*

#define [MM\_PEEK](#AAAAAAAAJL)  1

#define [MM\_POKE](#AAAAAAAAJM)  2

#define [MM\_LOAD\_FROM\_FILE](#AAAAAAAAJN)  3

#define [MM\_LOAD\_WID](#AAAAAAAAJO)  4

#define [MM\_DUMP\_TO\_FILE](#AAAAAAAAJP)  5

#define [MM\_DUMP\_INEVENT](#AAAAAAAAJQ)  6

#define [MM\_FILL](#AAAAAAAAJR)  7

#define [MM\_SYM\_LOOKUP](#AAAAAAAAJS)  8

#define [MM\_SYMTBL\_SAVE](#AAAAAAAAJT)  9

#define [MM\_EEPROMWRITE\_ENA](#AAAAAAAAJU)  10

#define [MM\_EEPROMWRITE\_DIS](#AAAAAAAAJV)  11

#define [MM\_NOOP](#AAAAAAAAJW)  12

#define [MM\_RESET](#AAAAAAAAJX)  13

### Macro Definition Documentation

#### #define MM\_BYTE\_BIT\_WIDTH  8

#### #define MM\_CLEAR\_ADDR  0

Used to clear out memory address variables.

#### #define MM\_CLEAR\_FNAME  '\0'

Used to clear out file name strings.

#### #define MM\_CLEAR\_PATTERN  0

Used to clear out fill and test patterns.

#### #define MM\_CLEAR\_SYMNAME  '\0'

Used to clear out symbol name strings.

#### #define MM\_DISABLE\_EEPROM\_WRITE\_CC  12

##### Purpose:

EEPROM Write Disable

##### Description

Disables writing to a specified EEPROM bank

##### Command Structure

[MM\_EepromWriteDisCmd\_t](#AAAAAAAAIX)

##### Command Verification

Successful execution of this command may be verified with the following telemetry:

- command counter will increment

- will be set to [MM\_EEPROMWRITE\_DIS](#AAAAAAAAJV)

The [MM\_EEPROM\_WRITE\_DIS\_INF\_EID](#AAAAAAAAGG) informational event message will be generated when the command is executed

##### Error Conditions

This command may fail for the following reason(s):

Command packet length not as expected

Non-success return status from PSP write disable

##### Evidence of failure may be found in the following telemetry:

- command error counter will increment

Error specific event message [MM\_LEN\_ERR\_EID](#AAAAAAAAEX)

Error specific event message [MM\_EEPROM\_WRITE\_DIS\_ERR\_EID](#AAAAAAAAGH)

##### Criticality

None

##### See also:

[MM\_ENABLE\_EEPROM\_WRITE\_CC](#AAAAAAAACG)

#### #define MM\_DUMP\_IN\_EVENT\_CC  7

##### Purpose:

Dump In Event Message

##### Description

Dumps up to [MM\_MAX\_DUMP\_INEVENT\_BYTES](#AAAAAAAADS) of memory in an event message

##### Command Structure

[MM\_DumpInEventCmd\_t](#AAAAAAAADQ)

##### Command Verification

Successful execution of this command may be verified with the following telemetry:

- command counter will increment

- will be set to [MM\_DUMP\_INEVENT](#AAAAAAAAJQ)

- will be set to the commanded memory type

- will be set to the fully resolved source memory address

- will be set to the number of bytes dumped

The [MM\_DUMP\_INEVENT\_INF\_EID](#AAAAAAAAET) informational event message will be generated with the dump data

##### Error Conditions

This command may fail for the following reason(s):

Command packet length not as expected

A symbol name was specified that can't be resolved

The address range fails validation check

The specified data size is invalid

The address and data size are not properly aligned

The specified memory type is invalid

##### Evidence of failure may be found in the following telemetry:

- command error counter will increment

Error specific event message [MM\_LEN\_ERR\_EID](#AAAAAAAAEX)

Error specific event message [MM\_SYMNAME\_ERR\_EID](#AAAAAAAAEZ)

Error specific event message [MM\_OS\_MEMVALIDATE\_ERR\_EID](#AAAAAAAAFE)

Error specific event message [MM\_DATA\_SIZE\_BYTES\_ERR\_EID](#AAAAAAAAFA)

Error specific event message [MM\_ALIGN32\_ERR\_EID](#AAAAAAAAFC)

Error specific event message [MM\_ALIGN16\_ERR\_EID](#AAAAAAAAFD)

Error specific event message [MM\_MEMTYPE\_ERR\_EID](#AAAAAAAAEY)

##### Criticality

It is the responsibility of the user to verify the *SrcSymAddress* , *NumOfBytes* , and *MemType* in the command. It is possible to generate a machine check exception when accessing I/O memory addresses/registers and other types of memory. The user is cautioned to use extreme care.

Note: Valid memory ranges are defined within a hardcoded structure contained in the PSP layer (CFE\_PSP\_MemoryTable) however, not every address within the defined ranges may be valid.

#### #define MM\_DUMP\_INEVENT  6

#### #define MM\_DUMP\_MEM\_TO\_FILE\_CC  6

##### Purpose:

Memory Dump To File

##### Description

Dumps the input number of bytes from processor memory to a file

##### Command Structure

[MM\_DumpMemToFileCmd\_t](#AAAAAAAAIT)

##### Command Verification

Successful execution of this command may be verified with the following telemetry:

- command counter will increment

- will be set to [MM\_DUMP\_TO\_FILE](#AAAAAAAAJP)

- will be set to the commanded memory type

- will be set to the fully resolved source memory address

- will be set to the number of bytes dumped

- will be set to the dump file name

The [MM\_DMP\_MEM\_FILE\_INF\_EID](#AAAAAAAAES) informational event message will be generated

##### Error Conditions

This command may fail for the following reason(s):

Command packet length not as expected

Command specified filename is invalid

A symbol name was specified that can't be resolved

#OS\_OpenCreate call fails

#CFE\_FS\_WriteHeader call fails

#OS\_close call fails

#OS\_write doesn't write the expected number of bytes or returns an error code

The #CFS\_ComputeCRCFromFile call fails

The address range fails validation check

The specified data size is invalid

The address and data size are not properly aligned

The specified memory type is invalid

##### Evidence of failure may be found in the following telemetry:

- command error counter will increment

Error specific event message [MM\_LEN\_ERR\_EID](#AAAAAAAAEX)

Error specific event message [MM\_CMD\_FNAME\_ERR\_EID](#AAAAAAAAFS)

Error specific event message [MM\_SYMNAME\_ERR\_EID](#AAAAAAAAEZ)

Error specific event message [MM\_OS\_CREAT\_ERR\_EID](#AAAAAAAAFK)

Error specific event message [MM\_CFE\_FS\_WRITEHDR\_ERR\_EID](#AAAAAAAAFW)

Error specific event message [MM\_OS\_CLOSE\_ERR\_EID](#AAAAAAAAFM)

Error specific event message [MM\_OS\_WRITE\_EXP\_ERR\_EID](#AAAAAAAAFP)

Error specific event message [MM\_CFS\_COMPUTECRCFROMFILE\_ERR\_EID](#AAAAAAAAFR)

Error specific event message [MM\_OS\_MEMVALIDATE\_ERR\_EID](#AAAAAAAAFE)

Error specific event message [MM\_DATA\_SIZE\_BYTES\_ERR\_EID](#AAAAAAAAFA)

Error specific event message [MM\_ALIGN32\_ERR\_EID](#AAAAAAAAFC)

Error specific event message [MM\_ALIGN16\_ERR\_EID](#AAAAAAAAFD)

Error specific event message [MM\_MEMTYPE\_ERR\_EID](#AAAAAAAAEY)

##### Criticality

It is the responsibility of the user to verify the *SrcSymAddress* , *NumOfBytes* , and *MemType* in the command. It is possible to generate a machine check exception when accessing I/O memory addresses/registers and other types of memory. The user is cautioned to use extreme care.

Note: Valid memory ranges are defined within a hardcoded structure contained in the PSP layer (CFE\_PSP\_MemoryTable) however, not every address within the defined ranges may be valid.

##### See also:

[MM\_LOAD\_MEM\_FROM\_FILE\_CC](#AAAAAAAAHN)

#### #define MM\_DUMP\_TO\_FILE  5

#### #define MM\_DWORD\_BIT\_WIDTH  32

#### #define MM\_EEPROM  2

EEPROM, requires special access for writes.

#### #define MM\_EEPROMWRITE\_DIS  11

#### #define MM\_EEPROMWRITE\_ENA  10

#### #define MM\_ENABLE\_EEPROM\_WRITE\_CC  11

##### Purpose:

EEPROM Write Enable

##### Description

Enables writing to a specified EEPROM bank

##### Command Structure

[MM\_EepromWriteEnaCmd\_t](#AAAAAAAAIW)

##### Command Verification

Successful execution of this command may be verified with the following telemetry:

- command counter will increment

- will be set to [MM\_EEPROMWRITE\_ENA](#AAAAAAAAJU)

The [MM\_EEPROM\_WRITE\_ENA\_INF\_EID](#AAAAAAAAGE) informational event message will be generated when the command is executed

##### Error Conditions

This command may fail for the following reason(s):

Command packet length not as expected

Non-success return status from PSP write enable

##### Evidence of failure may be found in the following telemetry:

- command error counter will increment

Error specific event message [MM\_LEN\_ERR\_EID](#AAAAAAAAEX)

Error specific event message [MM\_EEPROM\_WRITE\_ENA\_ERR\_EID](#AAAAAAAAGF)

##### Criticality

Extreme caution is advised in the use of this command. It is intended to be used only as a maintence tool for patching the default FSW image. This command will leave the EEPROM bank in a very vulnerable state. Once a patch has been completed the [MM\_DISABLE\_EEPROM\_WRITE\_CC](#AAAAAAAACF) command must be issued to protect the EEPROM bank from being inadvertently written.

##### See also:

[MM\_DISABLE\_EEPROM\_WRITE\_CC](#AAAAAAAACF)

#### #define MM\_FILL  7

#### #define MM\_FILL\_MEM\_CC  8

##### Purpose:

Memory Fill

##### Description

Reprograms processor memory with the fill pattern contained within the command message

##### Command Structure

[MM\_FillMemCmd\_t](#AAAAAAAAHI)

##### Command Verification

Successful execution of this command may be verified with the following telemetry:

- command counter will increment

- will be set to [MM\_FILL](#AAAAAAAAJR)

- will be set to the commanded memory type

- will be set to the fully resolved destination memory address

**MM\_FILLPATTERN -** will be set to the fill pattern used

- will be set to the number of bytes filled

The [MM\_FILL\_INF\_EID](#AAAAAAAAEL) informational event message will be generated when the command is executed

##### Error Conditions

This command may fail for the following reason(s):

Command packet length not as expected

A symbol name was specified that can't be resolved

The address range fails validation check

The specified data size is invalid

The address and data size are not properly aligned

The specified memory type is invalid

##### Evidence of failure may be found in the following telemetry:

- command error counter will increment

Error specific event message [MM\_LEN\_ERR\_EID](#AAAAAAAAEX)

Error specific event message [MM\_SYMNAME\_ERR\_EID](#AAAAAAAAEZ)

Error specific event message [MM\_OS\_MEMVALIDATE\_ERR\_EID](#AAAAAAAAFE)

Error specific event message [MM\_DATA\_SIZE\_BYTES\_ERR\_EID](#AAAAAAAAFA)

Error specific event message [MM\_ALIGN32\_ERR\_EID](#AAAAAAAAFC)

Error specific event message [MM\_ALIGN16\_ERR\_EID](#AAAAAAAAFD)

Error specific event message [MM\_MEMTYPE\_ERR\_EID](#AAAAAAAAEY)

##### Criticality

It is the responsibility of the user to verify the *DestSymAddress* , and *NumOfBytes* in the command. It is highly recommended to verify the success or failure of the memory fill. The fill may be verified by dumping memory and evaluating the dump contents. It is possible to destroy critical information with this command causing unknown consequences. In addition, it is possible to generate a machine check exception when accessing I/O memory addresses/registers and other types of memory. The user is cautioned to use extreme care.

Note: Valid memory ranges are defined within a hardcoded structure contained in the PSP layer (CFE\_PSP\_MemoryTable) however, not every address within the defined ranges may be valid.

#### #define MM\_LOAD\_FROM\_FILE  3

#### #define MM\_LOAD\_MEM\_FROM\_FILE\_CC  5

##### Purpose:

Memory Load From File

##### Description

Reprograms processor memory with the data contained within the given input file

##### Command Structure

[MM\_LoadMemFromFileCmd\_t](#AAAAAAAAIS)

##### Command Verification

Successful execution of this command may be verified with the following telemetry:

- command counter will increment

- will be set to [MM\_LOAD\_FROM\_FILE](#AAAAAAAAJN)

- will be set to the commanded memory type

- will be set to the fully resolved destination memory address

- will be set to the number of bytes loaded

- will be set to the load file name

The [MM\_LD\_MEM\_FILE\_INF\_EID](#AAAAAAAAEK) informational event message will be generated

##### Error Conditions

This command may fail for the following reason(s):

Command packet length not as expected

Command specified filename is invalid

#OS\_OpenCreate call fails

#OS\_close call fails

#OS\_read doesn't read the expected number of bytes

The #CFS\_ComputeCRCFromFile call fails

The computed CRC doesn't match the load file value

A symbol name was specified that can't be resolved

#CFE\_FS\_ReadHeader call fails

#OS\_read call fails

The address range fails validation check

The specified data size is invalid

The address and data size are not properly aligned

The specified memory type is invalid

##### Evidence of failure may be found in the following telemetry:

- command error counter will increment

Error specific event message [MM\_LEN\_ERR\_EID](#AAAAAAAAEX)

Error specific event message [MM\_CMD\_FNAME\_ERR\_EID](#AAAAAAAAFS)

Error specific event message [MM\_OS\_OPEN\_ERR\_EID](#AAAAAAAAFL)

Error specific event message [MM\_OS\_CLOSE\_ERR\_EID](#AAAAAAAAFM)

Error specific event message [MM\_OS\_READ\_EXP\_ERR\_EID](#AAAAAAAAFO)

Error specific event message [MM\_CFS\_COMPUTECRCFROMFILE\_ERR\_EID](#AAAAAAAAFR)

Error specific event message [MM\_LOAD\_FILE\_CRC\_ERR\_EID](#AAAAAAAAFF)

Error specific event message [MM\_SYMNAME\_ERR\_EID](#AAAAAAAAEZ)

Error specific event message [MM\_FILE\_LOAD\_PARAMS\_ERR\_EID](#AAAAAAAAFU)

Error specific event message [MM\_CFE\_FS\_READHDR\_ERR\_EID](#AAAAAAAAFV)

Error specific event message [MM\_OS\_READ\_ERR\_EID](#AAAAAAAAFN)

Error specific event message [MM\_OS\_MEMVALIDATE\_ERR\_EID](#AAAAAAAAFE)

Error specific event message [MM\_DATA\_SIZE\_BYTES\_ERR\_EID](#AAAAAAAAFA)

Error specific event message [MM\_ALIGN32\_ERR\_EID](#AAAAAAAAFC)

Error specific event message [MM\_ALIGN16\_ERR\_EID](#AAAAAAAAFD)

Error specific event message [MM\_MEMTYPE\_ERR\_EID](#AAAAAAAAEY)

##### Criticality

It is the responsibility of the user to verify the contents of the load file in the command. It is highly recommended to verify the success or failure of the memory load. The load may be verified by dumping memory and evaluating the dump contents. It is possible to destroy critical information with this command causing unknown consequences. In addition, it is possible to generate a machine check exception when accessing I/O memory addresses/registers and other types of memory. The user is cautioned to use extreme care.

Note: Valid memory ranges are defined within a hardcoded structure contained in the PSP layer (CFE\_PSP\_MemoryTable) however, not every address within the defined ranges may be valid.

##### See also:

[MM\_DUMP\_MEM\_TO\_FILE\_CC](#AAAAAAAADV)

#### #define MM\_LOAD\_MEM\_WID\_CC  4

##### Purpose:

Memory Load With Interrupts Disabled

##### Description

Reprogram processor memory with input data. Loads up to [MM\_MAX\_UNINTERRUPTIBLE\_DATA](#AAAAAAAAAV) data bytes into RAM with interrupts disabled

##### Command Structure

[MM\_LoadMemWIDCmd\_t](#AAAAAAAAHU)

##### Command Verification

Successful execution of this command may be verified with the following telemetry:

- command counter will increment

- will be set to [MM\_LOAD\_WID](#AAAAAAAAJO)

- will be set to the fully resolved destination memory address

- will be set to the number of bytes loaded

The [MM\_LOAD\_WID\_INF\_EID](#AAAAAAAAEJ) information event message will be generated when the command is executed

##### Error Conditions

This command may fail for the following reason(s):

Command packet length not as expected

A symbol name was specified that can't be resolved

The computed CRC doesn't match the command message value

The address range fails validation check

Invalid data size specified in command message

##### Evidence of failure may be found in the following telemetry:

- command error counter will increment

Error specific event message [MM\_LEN\_ERR\_EID](#AAAAAAAAEX)

Error specific event message [MM\_SYMNAME\_ERR\_EID](#AAAAAAAAEZ)

Error specific event message [MM\_LOAD\_WID\_CRC\_ERR\_EID](#AAAAAAAAFG)

Error specific event message [MM\_OS\_MEMVALIDATE\_ERR\_EID](#AAAAAAAAFE)

Error specific event message [MM\_DATA\_SIZE\_BYTES\_ERR\_EID](#AAAAAAAAFA)

##### Criticality

It is the responsibility of the user to verify the *DestSymAddress* , *NumOfBytes* , and *DataArray* contents in the command. It is highly recommended to verify the success or failure of the memory load. The load may be verified by dumping memory and evaluating the dump contents. It is possible to destroy critical information with this command causing unknown consequences. In addition, it is possible to generate a machine check exception when accessing I/O memory addresses/registers and other types of memory. The user is cautioned to use extreme care.

Note: Valid memory ranges are defined within a hardcoded structure contained in the PSP layer (CFE\_PSP\_MemoryTable) however, not every address within the defined ranges may be valid.

#### #define MM\_LOAD\_WID  4

#### #define MM\_LOOKUP\_SYM\_CC  9

##### Purpose:

Symbol Table Lookup

##### Description

Queries the system symbol table and reports the resolved address in telemetry and an informational event message

##### Command Structure

[MM\_LookupSymCmd\_t](#AAAAAAAAIU)

##### Command Verification

Successful execution of this command may be verified with the following telemetry:

- command counter will increment

- will be set to [MM\_SYM\_LOOKUP](#AAAAAAAAJS)

- will be set to the fully resolved memory address

##### Error Conditions

This command may fail for the following reason(s):

Command packet length not as expected

An empty string was specified as the symbol name

A symbol name was specified that can't be resolved

##### Evidence of failure may be found in the following telemetry:

- command error counter will increment

Error specific event message [MM\_LEN\_ERR\_EID](#AAAAAAAAEX)

Error specific event message [MM\_SYMNAME\_NUL\_ERR\_EID](#AAAAAAAAFZ)

Error specific event message [MM\_SYMNAME\_ERR\_EID](#AAAAAAAAEZ)

##### Criticality

None

##### See also:

[MM\_SYMTBL\_TO\_FILE\_CC](#AAAAAAAACK)

#### #define MM\_MEM16  4

Optional memory type that is only 16-bit read/write.

#### #define MM\_MEM32  5

Optional memory type that is only 32-bit read/write.

#### #define MM\_MEM8  3

Optional memory type that is only 8-bit read/write.

#### #define MM\_NOACTION  0

Used to clear out HK action variable.

#### #define MM\_NOMEMTYPE  0

Used to indicate that no memtype specified.

#### #define MM\_NOOP  12

#### #define MM\_NOOP\_CC  0

##### Purpose:

Noop

##### Description

Implements the Noop command that insures the MM task is alive

##### Command Structure

[MM\_NoArgsCmd\_t](#AAAAAAAAIR)

##### Command Verification

Successful execution of this command may be verified with the following telemetry:

- command counter will increment

The [MM\_NOOP\_INF\_EID](#AAAAAAAAEH) informational event message will be generated when the command is received

##### Error Conditions

This command may fail for the following reason(s):

Command packet length not as expected

##### Evidence of failure may be found in the following telemetry:

- command error counter will increment

Error specific event message [MM\_LEN\_ERR\_EID](#AAAAAAAAEX)

##### Criticality

None

##### See also:

[MM\_RESET\_CC](#AAAAAAAACJ)

#### #define MM\_NUM\_MEMTYPES  6

#### #define MM\_PEEK  1

#### #define MM\_PEEK\_CC  2

##### Purpose:

Memory Peek

##### Description

Reads 8,16, or 32 bits of data from any given input address

##### Command Structure

[MM\_PeekCmd\_t](#AAAAAAAADJ)

##### Command Verification

Successful execution of this command may be verified with the following telemetry:

- command counter will increment

- will be set to [MM\_PEEK](#AAAAAAAAJL)

- will be set to the commanded memory type

- will be set to the fully resolved destination memory address

- will be set to the byte size of the peek operation (1, 2, or 4)

The [MM\_PEEK\_BYTE\_INF\_EID](#AAAAAAAAEM) informational event message will be generated with the peek data if the data size was 8 bits

The [MM\_PEEK\_WORD\_INF\_EID](#AAAAAAAAEN) informational event message will be generated with the peek data if the data size was 16 bits

The [MM\_PEEK\_DWORD\_INF\_EID](#AAAAAAAAEO) informational event message will be generated with the peek data if the data size was 32 bits

##### Error Conditions

This command may fail for the following reason(s):

Command packet length not as expected

A symbol name was specified that can't be resolved

The specified data size is invalid

The specified memory type is invalid

The address range fails validation check

The address and data size are not properly aligned

##### Evidence of failure may be found in the following telemetry:

- command error counter will increment

Error specific event message [MM\_LEN\_ERR\_EID](#AAAAAAAAEX)

Error specific event message [MM\_SYMNAME\_ERR\_EID](#AAAAAAAAEZ)

Error specific event message [MM\_DATA\_SIZE\_BITS\_ERR\_EID](#AAAAAAAAFB)

Error specific event message [MM\_MEMTYPE\_ERR\_EID](#AAAAAAAAEY)

Error specific event message [MM\_OS\_MEMVALIDATE\_ERR\_EID](#AAAAAAAAFE)

Error specific event message [MM\_ALIGN16\_ERR\_EID](#AAAAAAAAFD)

Error specific event message [MM\_ALIGN32\_ERR\_EID](#AAAAAAAAFC)

##### Criticality

It is the responsibility of the user to verify the  *DestSymAddress*  and  *MemType*  in the command. It is possible to generate a machine check exception when accessing I/O memory addresses/registers and other types of memory. The user is cautioned to use extreme care.

Note: Valid memory ranges are defined within a hardcoded structure contained in the PSP layer (CFE\_PSP\_MemoryTable) however, not every address within the defined ranges may be valid.

##### See also:

[MM\_POKE\_CC](#AAAAAAAAHO)

#### #define MM\_POKE  2

#### #define MM\_POKE\_CC  3

##### Purpose:

Memory Poke

##### Description

Writes 8, 16, or 32 bits of data to any memory address

##### Command Structure

[MM\_PokeCmd\_t](#AAAAAAAAGZ)

##### Command Verification

Successful execution of this command may be verified with the following telemetry:

- command counter will increment

- will be set to [MM\_POKE](#AAAAAAAAJM)

- will be set to the commanded memory type

- will be set to the fully resolved source memory address

- will be set to the byte size of the poke operation (1, 2, or 4)

The [MM\_POKE\_BYTE\_INF\_EID](#AAAAAAAAEP) informational event message will be generated if the data size was 8 bits

The [MM\_POKE\_WORD\_INF\_EID](#AAAAAAAAEQ) informational event message will be generated if the data size was 16 bits

The [MM\_POKE\_DWORD\_INF\_EID](#AAAAAAAAER) informational event message will be generated if the data size was 32 bits

##### Error Conditions

This command may fail for the following reason(s):

Command packet length not as expected

A symbol name was specified that can't be resolved

The specified data size is invalid

The specified memory type is invalid

The address range fails validation check

The address and data size are not properly aligned

An EEPROM write error occured

##### Evidence of failure may be found in the following telemetry:

- command error counter will increment

Error specific event message [MM\_LEN\_ERR\_EID](#AAAAAAAAEX)

Error specific event message [MM\_SYMNAME\_ERR\_EID](#AAAAAAAAEZ)

Error specific event message [MM\_DATA\_SIZE\_BITS\_ERR\_EID](#AAAAAAAAFB)

Error specific event message [MM\_MEMTYPE\_ERR\_EID](#AAAAAAAAEY)

Error specific event message [MM\_OS\_MEMVALIDATE\_ERR\_EID](#AAAAAAAAFE)

Error specific event message [MM\_ALIGN16\_ERR\_EID](#AAAAAAAAFD)

Error specific event message [MM\_ALIGN32\_ERR\_EID](#AAAAAAAAFC)

Error specific event message [MM\_OS\_EEPROMWRITE8\_ERR\_EID](#AAAAAAAAFH)

Error specific event message [MM\_OS\_EEPROMWRITE16\_ERR\_EID](#AAAAAAAAFI)

Error specific event message [MM\_OS\_EEPROMWRITE32\_ERR\_EID](#AAAAAAAAFJ)

##### Criticality

It is the responsibility of the user to verify the *DestSymAddress* , *MemType* , and *Data* in the command. It is highly recommended to verify the success or failure of the memory poke. The poke may be verified by issuing a subsequent peek command and evaluating the returned value. It is possible to destroy critical information with this command causing unknown consequences. In addition, it is possible to generate a machine check exception when accessing I/O memory addresses/registers and other types of memory. The user is cautioned to use extreme care.

Note: Valid memory ranges are defined within a hardcoded structure contained in the PSP layer (CFE\_PSP\_MemoryTable) however, not every address within the defined ranges may be valid.

##### See also:

[MM\_PEEK\_CC](#AAAAAAAADX)

#### #define MM\_RAM  1

Normal RAM, no special access required.

#### #define MM\_RESET  13

#### #define MM\_RESET\_CC  1

##### Purpose:

Reset Counters

##### Description

Resets the MM housekeeping counters

##### Command Structure

[MM\_NoArgsCmd\_t](#AAAAAAAAIR)

##### Command Verification

Successful execution of this command may be verified with the following telemetry:

- command counter will be cleared

- command error counter will be cleared

The #MM\_RESET\_DBG\_EID informational event message will be generated when the command is executed

##### Error Conditions

This command may fail for the following reason(s):

Command packet length not as expected

##### Evidence of failure may be found in the following telemetry:

- command error counter will increment

Error specific event message [MM\_LEN\_ERR\_EID](#AAAAAAAAEX)

##### Criticality

None

##### See also:

[MM\_NOOP\_CC](#AAAAAAAACI)

#### #define MM\_SYM\_LOOKUP  8

#### #define MM\_SYMTBL\_SAVE  9

#### #define MM\_SYMTBL\_TO\_FILE\_CC  10

##### Purpose:

Save Symbol Table To File

##### Description

Saves the system symbol table to a file that can be transfered to the ground

##### Command Structure

[MM\_SymTblToFileCmd\_t](#AAAAAAAAIV)

##### Command Verification

Successful execution of this command may be verified with the following telemetry:

- command counter will increment

- will be set to [MM\_SYMTBL\_SAVE](#AAAAAAAAJT)

- will be set to the dump file name

The [MM\_SYMTBL\_TO\_FILE\_INF\_EID](#AAAAAAAAGA) informational event message will be generated when the command is executed

##### Error Conditions

This command may fail for the following reason(s):

Command packet length not as expected

An empty string was specified as the dump filename

The OSAL returns a status other than success to the command

##### Evidence of failure may be found in the following telemetry:

- command error counter will increment

Error specific event message [MM\_LEN\_ERR\_EID](#AAAAAAAAEX)

Error specific event message [MM\_SYMFILENAME\_NUL\_ERR\_EID](#AAAAAAAAGB)

Error specific event message [MM\_SYMTBL\_TO\_FILE\_FAIL\_ERR\_EID](#AAAAAAAAGC)

##### Note:

Dump filenames #OS\_MAX\_PATH\_LEN characters or longer are truncated

##### Criticality

None

##### See also:

[MM\_LOOKUP\_SYM\_CC](#AAAAAAAACH)

#### #define MM\_WORD\_BIT\_WIDTH  16

#### 

## fsw/src/mm\_utils.c File Reference

#include "mm\_app.h"

#include "mm\_utils.h"

#include "mm\_perfids.h"

#include "mm\_msgids.h"

#include "mm\_events.h"

#include "cfs\_utils.h"

#include "mm\_dump.h"

#include <string.h>

### Functions

* void [MM\_ResetHk](#AAAAAAAAJZ) (void)

*Reset housekeeping variables.*

* void [MM\_SegmentBreak](#AAAAAAAAKA) (void)

*Segment break.*

* bool [MM\_VerifyCmdLength](#AAAAAAAAKB) (const CFE\_MSG\_Message\_t \*MsgPtr, size\_t ExpectedLength)

*Verify command message length.*

* bool [MM\_VerifyPeekPokeParams](#AAAAAAAAKC) (cpuaddr Address, uint8 MemType, uint8 SizeInBits)

*Verify memory peek and poke parameters.*

* bool [MM\_VerifyLoadDumpParams](#AAAAAAAAKD) (cpuaddr Address, uint8 MemType, uint32 SizeInBytes, uint8 VerifyType)

*Verify memory load and dump parameters.*

### Variables

* [MM\_AppData\_t](#AAAAAAAACD) [MM\_AppData](#AAAAAAAAKE)

### Function Documentation

#### void MM\_ResetHk (void )

Reset housekeeping variables.

##### Description

Sets the local memory manager housekeeping variables to default values. This routine gets called before each command is processed to verify that all the variables are properly cleared

##### Assumptions, External Events, and Notes:

This function does not zero the command execution counter or the command error counter

#### void MM\_SegmentBreak (void )

Segment break.

##### Description

This routine gets called during each segment break in a load, dump, or memory fill operation and handles any processing that needs to be done during those breaks

##### Assumptions, External Events, and Notes:

None

#### bool MM\_VerifyCmdLength (const CFE\_MSG\_Message\_t \* *MsgPtr*, size\_t *ExpectedLength*)

Verify command message length.

##### Description

This routine will check if the actual length of a software bus command message matches the expected length and send an error event message if a mismatch occurs

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *MsgPtr* | A #CFE\_MSG\_Message\_t\* pointer that references the software bus message |
| in | *ExpectedLength* | The expected length of the message based upon the command code |

##### Returns:

Returns true if the length is as expected Returns false if the length is not as expected

##### See also:

[MM\_LEN\_ERR\_EID](#AAAAAAAAEX)

#### bool MM\_VerifyLoadDumpParams (cpuaddr *Address*, uint8 *MemType*, uint32 *SizeInBytes*, uint8 *VerifyType*)

Verify memory load and dump parameters.

##### Description

This routine will run various checks on the specified address, memory type, and data size (in bits) for a memory load or memory dump command.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *Address* | The source or destination address for the requested load or dump operation |
| in | *MemType* | The source or destination memory type for the requested load or dump operation |
| in | *SizeInBits* | The bit width for the requested load or dump operation |
| in | *VerifyType* | Flag indicating whether the requested operation is a load or a dump. |

##### Returns:

Returns true if all the parameter checks passed Returns false any parameter check failed

#### bool MM\_VerifyPeekPokeParams (cpuaddr *Address*, uint8 *MemType*, uint8 *SizeInBits*)

Verify memory peek and poke parameters.

##### Description

This routine will run various checks on the specified address, memory type, and data size (in bits) for a memory peek or memory poke command

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *Address* | The source or destination address for the requested peek or poke operation |
| in | *MemType* | The source or destination memory type for the requested peek or poke operation |
| in | *SizeInBits* | The bit width for the requested peek or poke operation |

##### Returns:

Returns true if all the parameter checks passed Returns false any parameter check failed

### Variable Documentation

#### [MM\_AppData\_t](#AAAAAAAACD) MM\_AppData

#### 

## fsw/src/mm\_utils.h File Reference

#include "mm\_msg.h"

### Functions

* void [MM\_ResetHk](#AAAAAAAAKG) (void)

*Reset housekeeping variables.*

* void [MM\_SegmentBreak](#AAAAAAAAKH) (void)

*Segment break.*

* bool [MM\_VerifyCmdLength](#AAAAAAAAKI) (const CFE\_MSG\_Message\_t \*MsgPtr, size\_t ExpectedLength)

*Verify command message length.*

* bool [MM\_VerifyPeekPokeParams](#AAAAAAAAKJ) (cpuaddr Address, uint8 MemType, uint8 SizeInBits)

*Verify memory peek and poke parameters.*

* bool [MM\_VerifyLoadDumpParams](#AAAAAAAAKK) (cpuaddr Address, uint8 MemType, uint32 SizeInBytes, uint8 VerifyType)

*Verify memory load and dump parameters.*

### Function Documentation

#### void MM\_ResetHk (void )

Reset housekeeping variables.

##### Description

Sets the local memory manager housekeeping variables to default values. This routine gets called before each command is processed to verify that all the variables are properly cleared

##### Assumptions, External Events, and Notes:

This function does not zero the command execution counter or the command error counter

#### void MM\_SegmentBreak (void )

Segment break.

##### Description

This routine gets called during each segment break in a load, dump, or memory fill operation and handles any processing that needs to be done during those breaks

##### Assumptions, External Events, and Notes:

None

#### bool MM\_VerifyCmdLength (const CFE\_MSG\_Message\_t \* *MsgPtr*, size\_t *ExpectedLength*)

Verify command message length.

##### Description

This routine will check if the actual length of a software bus command message matches the expected length and send an error event message if a mismatch occurs

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *MsgPtr* | A #CFE\_MSG\_Message\_t\* pointer that references the software bus message |
| in | *ExpectedLength* | The expected length of the message based upon the command code |

##### Returns:

Returns true if the length is as expected Returns false if the length is not as expected

##### See also:

[MM\_LEN\_ERR\_EID](#AAAAAAAAEX)

#### bool MM\_VerifyLoadDumpParams (cpuaddr *Address*, uint8 *MemType*, uint32 *SizeInBytes*, uint8 *VerifyType*)

Verify memory load and dump parameters.

##### Description

This routine will run various checks on the specified address, memory type, and data size (in bits) for a memory load or memory dump command.

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *Address* | The source or destination address for the requested load or dump operation |
| in | *MemType* | The source or destination memory type for the requested load or dump operation |
| in | *SizeInBits* | The bit width for the requested load or dump operation |
| in | *VerifyType* | Flag indicating whether the requested operation is a load or a dump. |

##### Returns:

Returns true if all the parameter checks passed Returns false any parameter check failed

#### bool MM\_VerifyPeekPokeParams (cpuaddr *Address*, uint8 *MemType*, uint8 *SizeInBits*)

Verify memory peek and poke parameters.

##### Description

This routine will run various checks on the specified address, memory type, and data size (in bits) for a memory peek or memory poke command

##### Assumptions, External Events, and Notes:

None

##### Parameters:

|  |  |  |
| --- | --- | --- |
| in | *Address* | The source or destination address for the requested peek or poke operation |
| in | *MemType* | The source or destination memory type for the requested peek or poke operation |
| in | *SizeInBits* | The bit width for the requested peek or poke operation |

##### Returns:

Returns true if all the parameter checks passed Returns false any parameter check failed

## fsw/src/mm\_verify.h File Reference

#include "mm\_mission\_cfg.h"

#include "mm\_platform\_cfg.h"

#include "stdint.h"

## fsw/src/mm\_version.h File Reference

### Macros

MM Version Identifiers#define [MM\_MAJOR\_VERSION](#AAAAAAAAGS)  2

#define [MM\_MINOR\_VERSION](#AAAAAAAAGT)  4

#define [MM\_REVISION](#AAAAAAAAGU)  2

### Macro Definition Documentation

#### #define MM\_MAJOR\_VERSION  2

#### #define MM\_MINOR\_VERSION  4

#### #define MM\_REVISION  2