

Flash Objects

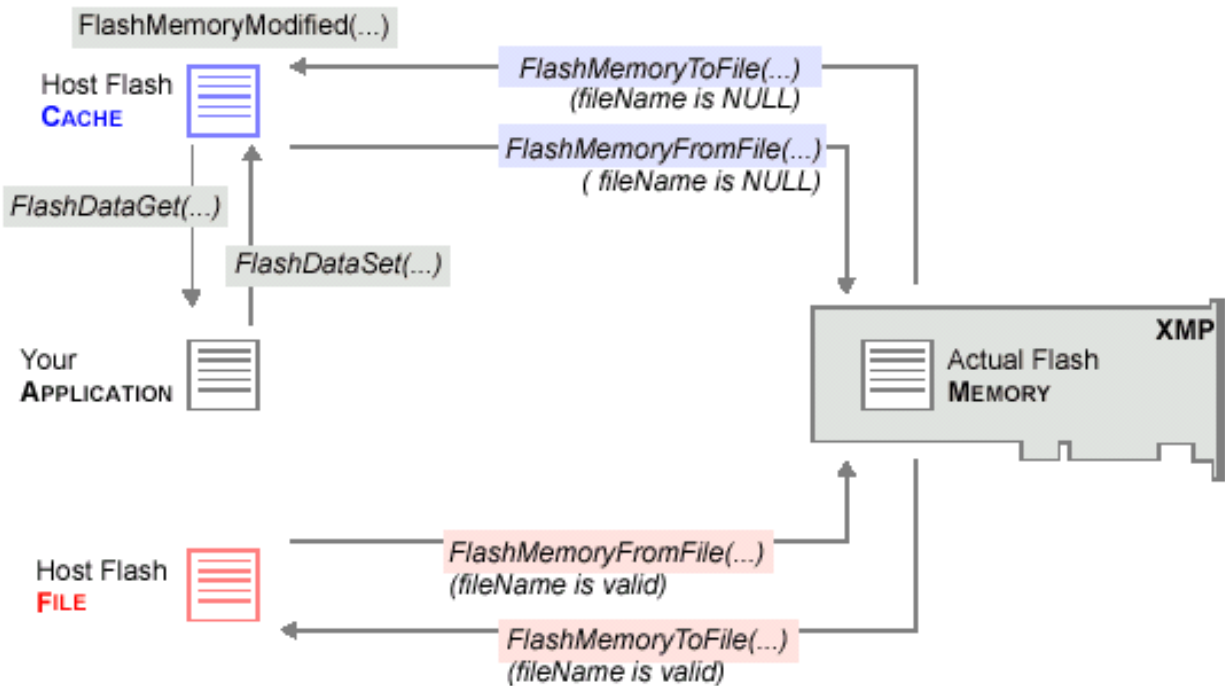
Introduction

A **Flash** object manages nonvolatile flash memory on the XMP/ZMP motion controllers. To optimize flash memory, a host-resident flash memory cache is used to provide faster writing performance.

After your application creates a Flash object (using `meiFlashCreate(...)`), the Flash object then creates a host-resident flash memory cache. `meiFlashCreate(...)` will create the flash object and initialize an internal (host-resident) cache with a copy of the flash on the board. All Flash functions (with the exception of `meiFlashMemoryFromFile(...)`) will modify the host resident cache. When you have finished modifying the host resident cache, use `meiFlashMemoryFromFile(fileName=NULL)` to write the host cache data to the board's flash memory. `meiFlashMemoryToFile(...)` can be used to copy the board's flash to a user specified file. (If the filename is NULL, the data is copied to the host resident cache.) Host resident cache is deleted when `meiFlashDelete(...)` is called.

Use the `meiFlashDataGet/Set(...)` methods to move data between your application and the flash cache. Use the `meiFlashMemoryToFile` and `meiFlashMemory/FromFile(...)` methods to move data between the flash cache (or file) and the actual flash memory. Typically, your application would:

1. Create a Flash object [using **meiFlashCreate(...)**].
2. Pass the **MEIFlash** handle to the **FlashConfig[Get/Set](...)** methods of the objects to be configured (which in turn call the **meiFlashData[Get/Set](...)** methods).
3. Write the flash cache to actual flash memory [using **meiFlashMemoryFromFile(...)**].
4. Delete the Flash object.



Methods

Create, Delete, Validate Methods

meiFlashCreate	Create Flash object
meiFlashDelete	Delete Flash object
meiFlashValidate	Validate Flash object

Configuration and Information Methods

meiFlashConfigGet	Copy flash config from cache to application memory
meiFlashConfigSet	Copy flash config from application memory to cache
meiFlashDataGet	Get count bytes of flash data memory and write them in application memory
meiFlashDataSet	Set count bytes of flash data memory using application memory

Memory Methods

meiFlashMemoryFromFileImage	Write actual flash memory using the binary image contained in filename
meiFlashMemoryFromFileType	Write actual flash memory using the binary image contained in filename
meiFlashMemoryGet	Copy count bytes of flash memory to application memory
meiFlashMemoryModified	Determine if flash cache has been modified
meiFlashMemorySet	Copy count bytes of application memory to flash memory
meiFlashMemoryFromFile	Write actual flash memory to cache or to file
meiFlashMemoryToFile	Save actual flash memory to cache or to file
meiFlashMemoryVerify	Verify actual flash memory

Relational Methods

meiFlashControl	Return handle of Control that is associated with Flash
---------------------------------	--

Data Types

[MEIFlashConfig](#)

[MEIFlashFileMaxNameChars](#)

[MEIFlashFileMaxChars](#)

[MEIFlashFileMaxPathChars](#)

[MEIFlashFiles](#)

[MEIFlashFileType](#)

[MEIFlashMessage](#)

[MEIFlashSection](#)

meiFlashCreate

Declaration `MEIFlash meiFlashCreate(MPIControl control)`

Required Header `stdmpi.h`

Description **FlashCreate** creates a Flash object and a host-resident copy of flash memory on motion controller *control* (called the flash cache). *FlashCreate* is the equivalent of a C++ constructor.



After FlashCreate is called, the flash cache is initialized with the contents of the actual flash memory.

Return Values

handle	to a Flash object
MPIHandleVOID	if the object could not be created

See Also [meiFlashDelete](#) | [meiFlashValidate](#)

meiFlashDelete

Declaration long [meiFlashDelete](#)([MEIFlash](#) **flash**)

Required Header stdmei.h

Description **FlashDelete** deletes a Flash object and invalidates its handle (*flash*).

FlashDelete is the equivalent of a C++ destructor.

Return Values

MPIMessageOK if *FlashDelete* successfully deletes a Flash object and invalidates its handle

See Also [meiFlashCreate](#) | [meiFlashValidate](#)

meiFlashValidate

Declaration long `meiFlashValidate`([MEIFlash](#) `flash`)

Required Header `stdmei.h`

Description `FlashValidate` validates the Flash object and its handle (*flash*).

Return Values

`MPIMessageOK` if Flash is a handle to a valid object.

See Also [meiFlashCreate](#) | [meiFlashDelete](#)

meiFlashDataGet

Declaration

```
long meiFlashDataGet(MEIFlash flash,
                    void *dst,
                    void *src,
                    long count)
```

Required Header

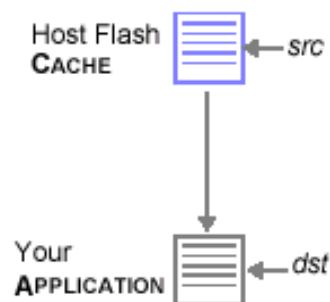
stdmei.h

Description

FlashDataGet gets *count* bytes of *flash* data memory starting at address *src* and puts (writes) them in application memory starting at address *dst*. The *src* pointer must point into the **MEIXmpData{...}** structure defined in *xmp.h* and be based on the firmware address (MEIXmpData *) returned by `mpiControlMemory(...)`.

Your application cannot access Flash memory directly; instead your application will access the host-resident flash memory cache maintained by *flash*.

meiFlashDataGet(...) reads from the flash cache and is called only by applications and utilities, while **meiFlashMemoryGet(...)** is a low-level method that reads directly from actual flash memory and is called primarily by other flash methods.



Return Values

MPIMessageOK	if <i>FlashDataGet</i> successfully gets count bytes of flash data memory writes them to application memory
---------------------	---

See Also

[mpiControlMemory](#) | [meiFlashMemoryGet](#) | [meiFlashDataSet](#)

meiFlashDataSet

Declaration

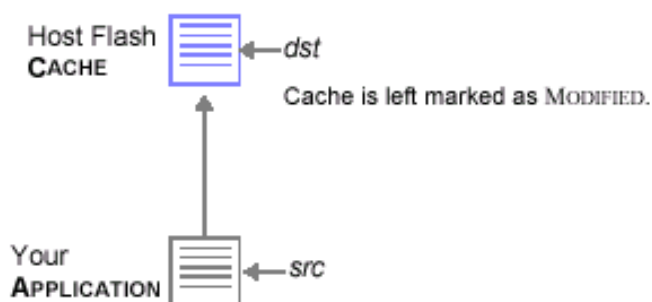
```
long meiFlashDataSet (MEIFlash flash,
                      void *dst,
                      void *src,
                      long count)
```

Required Header stdmei.h

Description **FlashDataSet** sets (writes) *count* bytes of *flash* data memory (starting at address *dst*) using application memory (starting at address *src*). The *dst* pointer must point into the **MEIXmpData{...}** structure defined in *xmp.h* and be based on the firmware address (MEIXmpData *) returned by mpiControlMemory(...).

Your application cannot access Flash memory directly; instead your application will access the host-resident flash memory cache maintained by flash.

mpiControlMemory(...) returns an external pointer that points to the MEIXmpBufferData{...} structure. You cannot use this external pointer with the FlashDataSet method to access flash data memory.



Return Values

MPIMessageOK if *FlashDataSet* successfully writes to the flash cache

See Also [mpiControlMemory](#) | [meiFlashDataSet](#)

meiFlashMemoryFromFileImage

Declaration

```
long meiFlashMemoryFromFileImage(MEIFlash flash,
                                   const char *fileImage,
                                   MEIFlashFileType *fileType);
```

Required Header stdmei.h

Description

FlashMemoryFromFileImage is used to erase and program controller flash memory. The values to be programmed are stored in the byte array `fileImage`. The parameter *fileType* is used by this method to determine the flash sector and size of the flash memory region to be programmed.

Supported values for *fileType* are:

- **MEIFlashFileTypeCode**, Code
- **MEIFlashFileTypeDataInt**, Internal Data
- **MEIFlashFileTypeDataExt**, External Data
- **MEIFlashFileTypeSynqNet**, SynqNet
- **MEIFlashFileTypeCodeAndData**, Code, Internal and External Data.
- **MEIFlashFileTypeFPGA0**, FPGA (Rincon)
- **MEIFlashFileTypeALL**, Code, Internal Data, External Data, and FPGA.

This method exists primarily for compatibility with older software. For new designs, it is recommended that you use the the [meiFlashMemoryFromFileType\(\)](#) method instead.

flash	a handle to a flash object
*fileImage	a pointer to data in memory to be copied to flash.
*fileType	where the fileImage will be copied to in flash.

Return Values

MPIMessageOK	if <i>FlashMemoryFromFileImage</i> successfully erases/programs the controller flash memory.
MEIFlashMessageFLASH_INVALID	if the flash handle is invalid.
MPIMessageNO_MEMORY	the system could not allocate enough memory for temporary storage of the flash image.

MPIMessageTIMEOUT	the flash could not be programmed.
MEIFlashMessageFLASH_WRITE_ERROR	the programmed flash did not match the flash image in verification.

See Also [meiFlashMemoryFromFileType](#)

meiFlashMemoryGet

Declaration

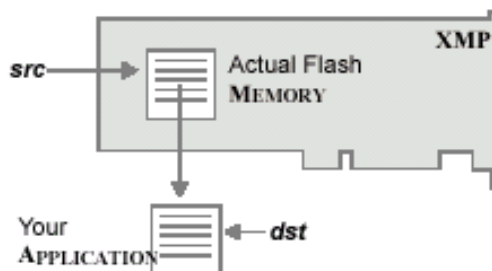
```
long meiFlashMemoryGet (MEIFlash flash,
                        void *dst,
                        void *src,
                        long count)
```

Required Header stdmei.h

Description **FlashMemoryGet** copies *count* bytes of actual flash memory (*flash*, starting at address *src*) to application memory (starting at address *dst*).

You should calculate the *src* pointer by considering flash memory as a stream of bytes, because FlashMemoryGet will adjust the pointer for the actual type of flash memory.

meiFlashMemoryGet(...) is a low-level method that reads directly from actual flash memory and is called primarily by other flash methods, while **meiFlashDataGet(...)** reads from the flash cache and is called only by applications and utilities.



XMP Only

flash memory is of type *unsigned long **.

Return Values

MPIMessageOK if *FlashMemoryGet* successfully copies flash memory to application memory

See Also [meiFlashMemorySet](#)

meiFlashMemorySet

Declaration

```
long meiFlashMemorySet (MEIFlash flash,
                          void *dst,
                          void *src,
                          long count)
```

Required Header `stdmei.h`

Description [FlashMemorySet](#) copies application memory (starting at address *src*) to *count* bytes of flash memory (*flash*, starting at address *dst*).

You should calculate the *dst* pointer by considering flash memory as a stream of bytes, because FlashMemoryGet will adjust the pointer for the actual type of flash memory.

[XMP Only](#)

flash memory is of type *unsigned long **.

Return Values

MPIMessageOK if *FlashMemorySet* successfully copies application memory to flash memory

See Also [meiFlashMemoryGet](#)

meiFlashControl

Declaration `MPIControl meiFlashControl(MEIFlash flash)`

Required Header `stdmei.h`

Description **FlashControl** returns a handle to the motion controller (Control object) that a Flash object (*flash*) is associated with.

Return Values

handle	to a Control object that a Flash object is associated with
MPIHandleVOID	if the Flash object is invalid

See Also

MEIFlashConfig

MEIFlashConfig

```
typedef struct MEIFlashConfig {
    long                wordSize;
    long                sectorSize;
    long                extMemSize;
    MEIFlashSection    all;
    MEIFlashSection    code;
    MEIFlashSection    codeBoot0;
    MEIFlashSection    codeBoot;
    MEIFlashSection    codeMain;
    MEIFlashSection    data;
    MEIFlashSection    dataExt;
    MEIFlashSection    synqNet;
    MEIFlashSection    FPGA0;
} MEIFlashConfig;
```

Description

The **FlashConfig** structure contains the flash configuration parameters. This data is stored in the host's memory and is internally used by the MPI library to manage flash reads/writes. Typically, applications do not need to access the flash configurations.

WARNING:

This data structure is for MEI internal purposes only and should not be configured by a customer.

wordSize	The size of a flash word in bytes.
sectorSize	The size of a flash sector in bytes.
extMemSize	The size of the controller's external memory in bytes.
all	The flash section parameters for the controller's code, data, and local FPGA images.
code	The flash section parameters for the controller's boot and main code.
codeBoot0	The flash section parameters for the controller's boot and main code. (ZMP)
codeBoot	The flash section parameters for the controller's boot code.
codeMain	The flash section parameters for the controller's main code.
data	The flash section parameters for the controller's internal data memory.
dataExt	The flash section parameters for the controller's external data memory.
synqNet	This is for the controller's synqnet data memory.
FPGA0	The flash section parameters for the local FPGA image number 0.

See Also

[MEIFlashSection](#) | [MEIFlashFileType](#) | [meiFlashConfigGet](#) | [meiFlashConfigSet](#)

MEIFlashFileMaxNameChars

Declaration `#define MEIFlashFileMaxNameChars (12) /*8.3 format */`

Required Header `stdmei.h`

Description The **FlashFileMaxNameChars** define is used internally by the MPI when linking a series of strings to create the full path name of the flash file. If the user tries to use a file with more than 12 characters (8.3 format), then the MPI will not be able to create the correct filename and may result in problems when opening the file.

See Also [MEIFlashFileMaxChars](#) | [MEIFlashFileMaxPathChars](#)

MEIFlashFileMaxChars

Declaration `#define MEIFlashFileMaxChars (120)`

Required Header `stdmei.h`

Description The MPI has to create filenames with the full path in order to open the flash files. **FlashFileMaxChars** is an arbitrary size that is used by the MPI to allocate memory space for the filename. Full path filenames should not exceed 120 characters.

See Also [MEIFlashFileMaxNameChars](#) | [MEIFlashFileMaxPathChars](#)

MEIFlashFileMaxPathChars

Declaration

```
#define MEIFlashFileMaxPathChars ( MEIFlashFileMaxChars -  
MEIFlashFileMaxNameChars )
```

Required Header `stdmei.h`

Description [FlashFileMaxPathChars](#) is used internally by the MPI when creating full path filenames. This is the number of characters available for the path, not including the actual filename.

See Also [MEIFlashFileMaxNameChars](#) | [MEIFlashFileMaxChars](#)

MEIFlashFiles

MEIFlashFiles

```
typedef struct MEIFlashFiles {  
    char binFile[MEIFlashFileMaxChars];  
    char FPGAFile[MEIXmpFlashMaxFPGAFiles][MEIFlashFileMaxChars];  
} MEIFlashFiles;
```

Description

The **FlashFiles** structure specifies the binary files to be downloaded to the controller.

binFile	A controller firmware filename. The firmware file contains configuration data and executable code. The firmware file format is binary.
FPGAFile	An array of FPGA filenames. The FPGA file contains the binary image that is loaded into the FPGA component. Only the appropriate FPGA file can be loaded.

See Also

[meiFlashMemoryFromFile](#)

MEIFlashFileType

MEIFlashFileType

```
typedef enum {
    MEIFlashFileTypeNONE = 0,
    MEIFlashFileTypeCode,
    MEIFlashFileTypeDataInt,
    MEIFlashFileTypeDataExt,
    MEIFlashFileTypeSynqNet,
    MEIFlashFileTypeCodeAndData,
    MEIFlashFileTypeFPGA0,
    MEIFlashFileTypeALL      /* Loads Code and all FPGAs
                               (for .bin files that include
                               the FPGA images) */
} MEIFlashFileType;
```

Description **FlashFileType** is an enumeration of file types. Each file type contains code or data for the controller's flash memory. This enumeration is used to specify what code/data is to be copied to flash memory with `meiFlashMemoryFromFileType(...)` or copied from flash memory with `meiFlashMemoryToFile(...)`.

MEIFlashFileTypeCode	Controller processor code only.
MEIFlashFileTypeDataInt	Controller internal data only.
MEIFlashFileTypeDataExt	Controller external data only.
MEIFlashFileTypeSynqNet	Used for flashing the synqNet page of flash.
MEIFlashFileTypeCodeAndData	Controller processor code and data.
MEIFlashFileTypeFPGA0	Local FPGA image number 0.
MEIFlashFileTypeALL	All code, data, and FPGA images.

See Also [meiFlashMemoryFromFileType](#) | [meiFlashMemoryToFile](#) | [meiFlashMemoryVerify](#)

MEIFlashMessage

MEIFlashMessage

```
typedef enum {
    MEIFlashMessageFLASH_INVALID,
    MEIFlashMessageFLASH_VERIFY_ERROR,
    MEIFlashMessageFLASH_WRITE_ERROR,
    MEIFlashMessagePATH,
    MEIFlashMessageNETWORK_TOPOLOGY_ERROR,
} MEIFlashMessage;
```

Description [FlashMessage](#) lists the error messages returned by the MEIFlash module.

MEIFlashMessageFLASH_INVALID

The flash object is not valid. This message code is returned by [meiFlashMemoryFromFile\(...\)](#), [meiFlashMemoryFromFileType\(...\)](#), [meiFlashMemoryToFile\(...\)](#), or [meiFlashMemoryVerify\(...\)](#) if the flash object's memory cache has not been allocated. The flash memory cache is allocated in [mpiFlashCreate\(...\)](#). To prevent this problem, create flash objects with [mpiFlashCreate\(...\)](#) and check the return value.

MEIFlashMessageFLASH_VERIFY_ERROR

The flash memory verify failed. This message code is returned by [meiFlashMemoryVerify\(...\)](#) if the read from flash memory does not match the read from the file. This indicates that the specified flash file is different from the flash memory. This message code can also be returned from [meiFlashMemoryFromFile\(...\)](#), since it calls [meiFlashMemoryVerify\(...\)](#). To correct this problem, first check that the specified file is the one you intended. If the specified file is correct, use [meiFlashMemoryFromFile\(...\)](#) to download the file.

MEIFlashMessageFLASH_WRITE_ERROR

The flash memory write failed. This message code is returned by [meiFlashMemoryFromFile\(...\)](#) or [meiFlashMemoryFromFileType\(...\)](#) if the flash memory write fails. This indicates a problem in the flash memory component or subsystem.

MEIFlashMessagePATH

The flash file path is too long. This message code is returned by [meiFlashMemoryFromFile\(...\)](#) if the file path is longer than `MEIFlashFileMaxPathChars`. To correct this problem, use a shorter path.

MEIFlashMessageNETWORK_TOPOLOGY_ERROR

The network topology has not been saved to flash. This message code is returned by object flash config set methods if the network topology has not been previously saved to flash. This message code serves as a warning of a potential safety problem. Saving object configurations to flash will cause those values to be sent to nodes during the next network initialization, even if the flash configurations are not compatible with the network topology. If the network topology is saved to flash, then the controller will automatically verify the topology before sending object configuration values to the nodes.

See Also [MEIFlash](#)

MEIFlashSection

MEIFlashFiles

```
typedef struct MEIFlashSection {
    unsigned char    *address;
    long             size;
    long             sectorIndex;
} MEIFlashSection;
```

Description

The **FlashSection** structure contains the flash configuration parameters for a specified section.

*address	A pointer to a flash sector address.
size	Length of the flash memory in bytes.
sectorIndex	The flash sector number.

See Also

[MEIFlashConfig](#) | [meiFlashConfigGet](#) | [meiFlashConfigSet](#)