Flash Objects

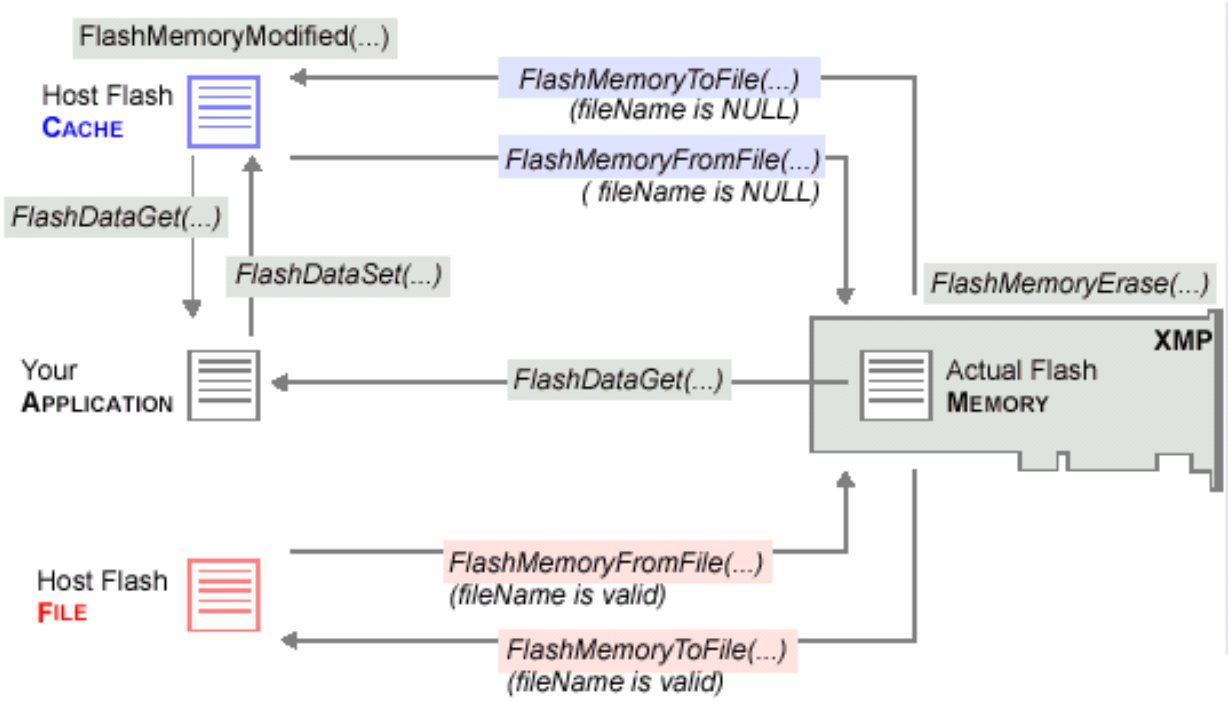
Introduction

A **Flash** object manages nonvolatile flash memory on the XMP motion controller. Because all of flash memory must be written as one contiguous piece (which takes a lot of time), and a host-resident *flash memory cache* is used to provide faster writing performance, and to also reduce the number of write cycles performed.

After your application creates a Flash object (**using meiFlashCreate(...)**), the Flash object then creates a host-resident flash memory cache, initializing it using a copy of actual flash memory [by calling **meiFlashMemoryToFile(fileName=Null)**]. Using the data structures defined in xmp.h and pointers to those structures (returned by the **mpiControlMemory(...)** method), your application can then use the **meiFlashData[Get/Set](...)** methods to access this memory cache. Note that the **meiFlashDataSet(...)** method updates the flash cache, and doesn't update the actual flash memory. To write the flash memory cache to actual flash memory, your application must call **meiFlashMemoryFromFile(fileName=Null)**.

Use the **FlashDataGet/Set(...)** methods to move data between your application and the flash cache. Use the **FlashMemoryTo/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

meiFlash**Create**	Create Flash object
meiFlash**Delete**	Delete Flash object
meiFlash**Validate**	Validate Flash object

Configuration and Information Methods

meiFlash**ConfigGet**	Copy flash config from cache to application memory
meiFlash**ConfigSet**	Copy flash config from application memory to cache
meiFlash**DataGet**	Get count bytes of flash data memory and write them in application memory
meiFlash**DataSet**	Set count bytes of flash data memory using application memory

Memory Methods

meiFlash**MemoryErase**	Erase actual flash memory on XMP
meiFlash**MemoryFromFile**	Write actual flash memory using cache or file
meiFlash**MemoryFromFileType**	Write actual flash memory using the binary image contained in filename
meiFlash**MemoryGet**	Copy count bytes of flash memory to application memory
meiFlash**MemoryModified**	Determine if flash cache has been modified
meiFlash**MemorySet**	Copy count bytes of application memory to flash memory
meiFlash**MemoryToFile**	Save actual flash memory to cache or to file
meiFlash**MemoryVerify**	Compare actual flash memory to cache or to file

Relational Methods

meiFlash**Control**	Return handle of Control that is associated with Flash

Data Types

MEIFlash**Files**
MEIFlash**Message**

meiFlashCreate

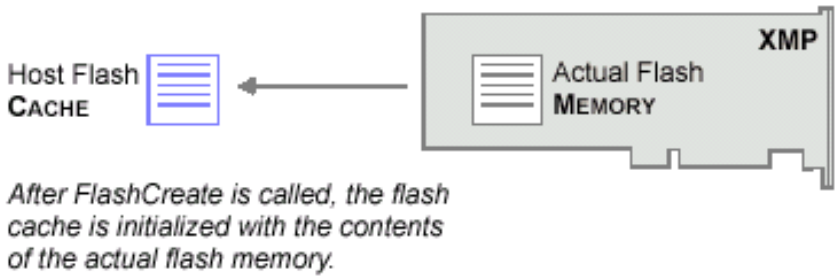
Declaration

```
const 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.



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

meiFlashConfigGet

Declaration

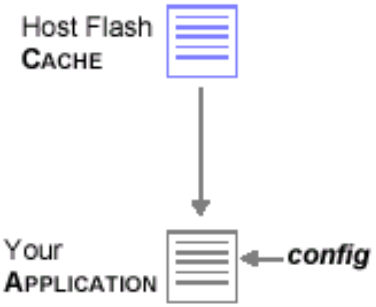
```
long meiFlashConfigGet(MEIFlash        flash,
                       MEIFlashConfig  *config)
```

Required Header

stdmei.h

Description

FlashConfigGet gets the flash configuration and writes it into the structure pointed to by ***config***. The flash configuration includes data about the actual flash memory device (its type and size).



Return Values	
MPIMessageOK	if *FlashConfigGet* successfully gets the flash configuration and writes it into the structure

See Also

meiFlashConfigSet

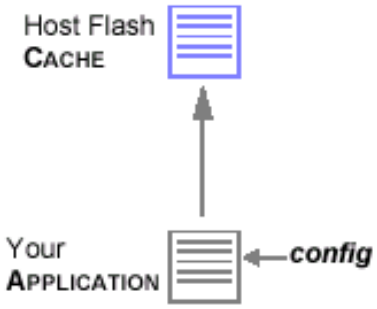
meiFlashConfigSet

Declaration

```
long meiFlashConfigSet(MEIFlash        flash,
                       MEIFlashConfig  *config)
```

Required Header stdmei.h

Description **FlashConfigSet** sets (reads) the control configuration from the structure pointed to by ***config***.



Return Values	
MPIMessageOK	if *FlashConfigSet* successfully reads the control configuration from the structure

See Also meiFlashConfigGet

meiFlashDataGet

Declaration

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

Required Header

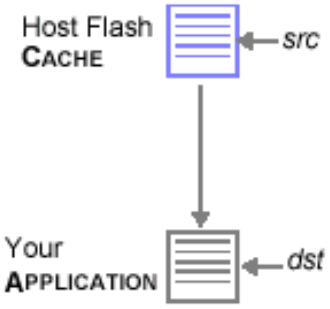
stdmei.h

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 *FlashDataGet* 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

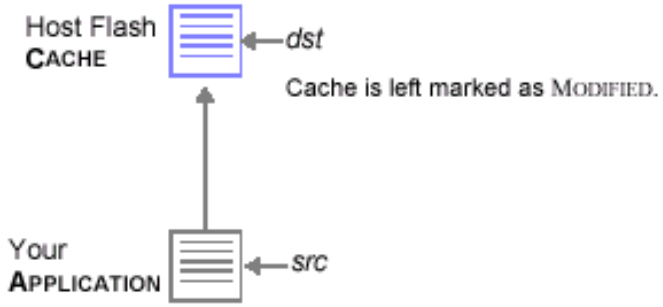
stdmei.h

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 | meiFlashDataGet

meiFlashMemoryErase

Declaration

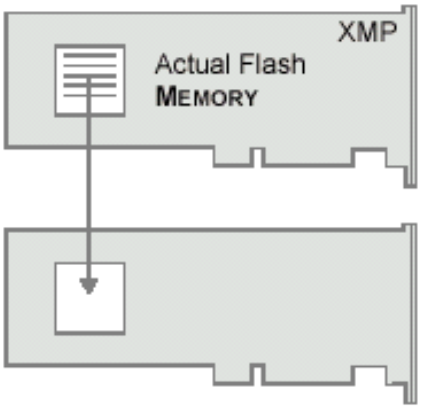
```
long meiFlashMemoryErase(MEIFlash flash,
                         long     erase,
                         long     verify)
```

Required Header

stdmei.h

Description

FlashMemoryErase erases actual flash memory, and also verifies that actual flash memory is erased.



flash	a handle to a Flash object
erase	a value, either zero or non-zero
verify	a value, either zero or non-zero

Return Values

MPIMessageOK	if *FlashMemoryErase* successfully erases the flash memory.

See Also

meiFlashCreate

meiFlashMemoryFromFile

Declaration

```
long meiFlashMemoryFromFile(MEIFlash       flash,
                            MEIFlashFiles  *filesIn,
                            MEIFlashFiles  *filesOut)
```

Required Header stdmei.h

Description

FlashMemoryFromFile writes actual flash memory using the binary image contained in fileName, or using the flash memory cache.

Also, calling the meiFlashMemoryFromFile(...) method resets the flash cache modified indication to FALSE.

Return Values	
MPIMessageOK	if *FlashMemoryFromFile* successfully writes to actual flash memory using the file or using flash cache

See Also meiFlashMemoryFromFileType

meiFlashMemoryFromFileType

Declaration

```
long meiFlashMemoryFromFileType(MEIFlash          flash,
                                const char        *fileName,
                                MEIFlashFileType  fileType)
```

Required Header

stdmei.h

Description

FlashMemoryFromFileType writes actual flash memory using the binary image contained in ***fileName***, or using the flash memory cache.

flash	a handle to a Flash object
***filename**	a string
fileType	an enumeration corresponding to the flash file types

Return Values

MPIMessageOK	if *FlashMemoryFromFileType* successfully copies the data from the file to the flash.

See Also

meiFlashCreate | meiFlashMemoryFromFile

meiFlashMemoryGet

Declaration

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

Required Header

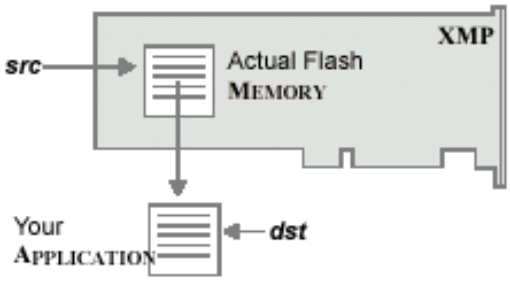
stdmei.h

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

meiFlashMemoryModified

Declaration

```
long meiFlashMemoryModified(MEIFlash flash,
                            long     *modified)
```

Required Header

stdmei.h

Description

FlashMemoryModified determines if the flash memory cache has been modified. Note that unless meiFlashDataSet(...) has been called previously, the meiFlashMemoryModified(...) method will always return False, regardless of whether the cache has been modified or not modified.

If the "flash cache"	*Then*
has been modified	*FlashMemoryModified* writes True to the location pointed to by modified
has not been modified	*FlashMemoryModified* writes False to the location pointed to by modified

Return Values	
MPIMessageOK	if *FlashMemoryModified* successfully determines if the flash cache has been modified

See Also

meiFlashDataSet

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

meiFlashMemoryToFile

Declaration

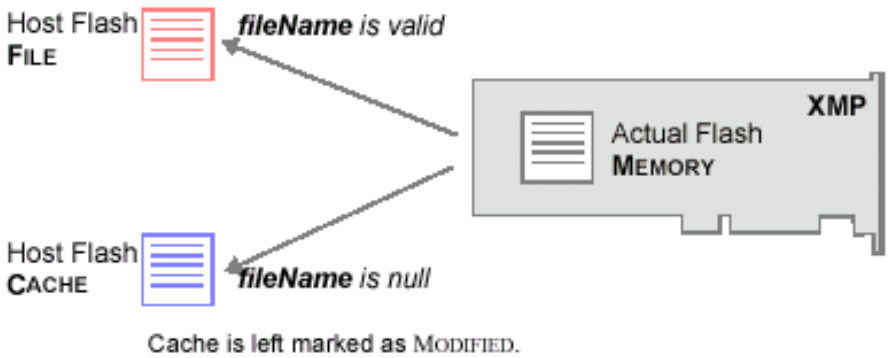
```
long meiFlashMemoryToFile(MEIFlash            flash,
                          const char          *fileName,
                          MEIFlashFileType    fileType)
```

Required Header

stdmei.h

Description

FlashMemoryToFile saves actual ***flash*** memory to a binary image contained in ***fileName***, or to the flash memory cache.



Return Values

MPIMessageOK	if *FlashMemoryToFile* successfully saves actual flash memory to file or to cache

See Also

meiFlashMemoryVerify

Declaration

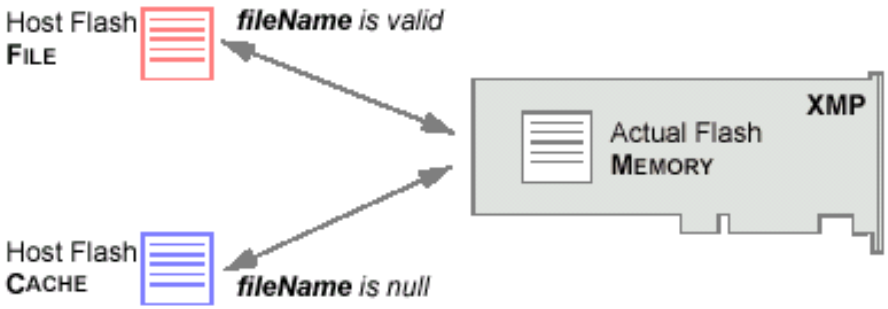
```
long meiFlashMemoryVerify(MEIFlash      flash,
                          const char    *fileName)
```

Required Header

stdmei.h

Description

FlashMemoryVerify compares actual ***flash*** memory to the binary image contained in ***fileName***, or to the flash memory cache (if ***fileName*** is Null). Every word of flash memory (including code and data) is compared with the firmware bin file (or cache if the ***fileName*** is Null). The firmware version number is contained in flash data memory, so even if the only difference is the firmware version number, flash verification will fail. Flash verification will fail if *any* word is different.



Cache is left marked as MODIFIED.

Return Values

MPIMessageOK	if actual flash memory is identical to the binary image in fileName or in cache
MPIMessageNo_Memory	if memory allocation error occurred
MPIMessageFILE_OPEN_ERROR	if the file (***fileName***) was not found
MPIMessageFILE_READ_ERROR	if an error occurred while reading the file (***fileName***)
MPIMessageFILE_CLOSE_ERROR	if an error occurred whileclosing the file (***fileName***)
MPIFlashMessageFLASH_READ_ERROR	if differences were detected during flash validation

See Also

meiFlashControl

Declaration

```
const 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

MEIFlashFiles

MEIFlashFiles

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

Description

FlashFiles is a configuration structure that specifies what files should be copied to the flash memory of the controller.

binFile	A string buffer that holds the filename of the .bin file to be flashed.
FPGAFile	An array of string buffers that holds the filenames of the FPGA files to be flashed.

See Also

MEIFlash

MEIFlashMessage

MEIFlashMessage

```
typedef enum {

    MEIFlashMessageFLASH_INVALID,
    MEIFlashMessageFLASH_READ_ERROR,
    MEIFlashMessageFLASH_WRITE_ERROR,
    MEIFlashMessagePATH,
} MEIFlashMessage;
```

Description

FlashMessage lists the error messages returned by the MEIFlash module.

MEIFlashMessageFLASH_INVALID	The flash object passed to meiFlash…() methods was invalid.
MEIFlashMessageFLASH_READ_ERROR	The flash object could not successfully read data.
MEIFlashMessageFLASH_WRITE_ERROR	The flash object could not successfully write data.
MEIFlashMessagePATH	The flash file path is too long.

See Also

MEIFlash