Filter Objects

Introduction

A Filter object manages a single filter on a controller. It represents the control algorithm used to control a motor in a closed-loop system. The Filter contains an algorithm, a set of coefficients, inputs, and an output. Its primary responsibility is to take the difference between the command and actual positions and then calculate the output based on the control algorithm and coefficients.

For simple systems, there is a one-to-one relationship between the Axis, Filter, and Motor objects.

Methods

Create, Delete, Validate Methods

mpiFilterCreate Create Filter object
mpiFilterDelete Delete Filter object
mpiFilterValidate Validate Filter object

Configuration and Information Methods

mpiFilterConfigGet Get Filter configuration
mpiFilterConfigSet Set Filter configuration
mpiFilterFlashConfigGet Get flash configuration for Filter
mpiFilterFlashConfigSet Set flash configuration for Filter
mpiFilterGainGet Get gain coefficients
mpiFilterGainSet Set current gain index
mpiFilterGainIndexGet Get current gain index
mpiFilterGainIndexSet Set current gain index
mpiFilterStatus Get Filter’s status

Memory Methods

mpiFilterMemory Get address to Filter memory
mpiFilterMemoryGet Copy data from Filter memory to application memory
mpiFilterMemorySet Copy data from application memory to Filter memory

Relational Methods

mpiFilterAxisMapGet Get object map of axes associated with Filter
mpiFilterAxisMapSet Set axes associated with Filter
mpiFilterControl Return handle of Control that is associated with Filter
mpiFilterMotorMapGet Get object map of Motors associated with Filter
mpiFilterMotorMapSet Set Motors to be associated with Filter
mpiFilterNumber Get index of Filter (for Control list)

Action Methods

mpiFilterIntegratorReset Reset the integrators of filter.
Data Types

- MPIFilterCoeff
- MPIFilterConfig / MEIFilterConfig
- MPIFilterGain
- MEIFilterGainIndex
- MEIFilterGainPID
- MEIFilterGainPIDCoeff
- MEIFilterGainPIV
- MEIFilterGainPIVCoeff
- MEIFilterGainSercos
- MEIFilterGainSercosCoeff
- MPIFilterMessage

Constants

- MPIFilterCoeffCOUNT_MAX
- MPIFilterGainCOUNT_MAX

Copyright © 2002
Motion Engineering
**mpiFilterCreate**

**Declaration**

```cpp
const MPIFilter mpiFilterCreate(MPIControl control, long number)
```

**Required Header**

`stdmpi.h`

**Description**

`FilterCreate` creates a Filter object associated with a filter (`number`), that is located on a motion controller (`control`).

`FilterCreate` is the equivalent of a C++ constructor.

**Return Values**

<table>
<thead>
<tr>
<th>Return Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>handle</td>
<td>to an Filter object</td>
</tr>
<tr>
<td>MPIHandleVOID</td>
<td>if the Filter object could not be created</td>
</tr>
</tbody>
</table>

**See Also**

`mpiFilterDelete` | `mpiFilterValidate`
mpiFilterDelete

Declaration
long mpiFilterDelete(MPIFilter filter)

Required Header stdmpi.h

Description
FilterDelete deletes a Filter object and invalidates its handle (filter). FilterDelete is the equivalent of a C++ destructor.

Return Values
MPIMessageOK if FilterDelete successfully deletes a Filter object and invalidates its handle

See Also
mpiFilterCreate | mpiFilterValidate

mpiFilterValidate

Declaration

long mpiFilterValidate(MPIFilter filter)

Required Header
stdmpi.h

Description

FilterValidate validates the Filter object and its handle (filter).

Return Values

MPIMessageOK if Filter is a handle to a valid object.

See Also

mpiFilterCreate | mpiFilterDelete
**mpiFilterConfigGet**

**Declaration**

```c
long mpiFilterConfigGet (MPIFilter filter,
                        MPIFilterConfig *config,
                        void *external)
```

**Required Header**

`stdmpi.h`

**Description**

`FilterConfigGet` gets a Filter’s (`filter`) configuration and writes it into the structure pointed to by `config`, and also writes it into the implementation-specific structure pointed to by `external` (if `external` is not NULL).

The Filter’s configuration information in `external` is in addition to the Filter’s configuration information in `config`, i.e., the Filter’s configuration information in `config` and in `external` is not the same information. Note that `config` or `external` can be NULL (but not both NULL).

**XMP Only**

`external` either points to a structure of type `MEIFilterConfig` or is NULL.

**Return Values**

- `MPIMessageOK` if `FilterConfigGet` successfully writes the Filter’s configuration to the structure(s)

**See Also**

`mpiFilterConfigSet` | `MEIFilterConfig`
### mpiFilterConfigSet

#### Declaration

```c
long mpiFilterConfigSet(MPIFilter filter, MPIFilterConfig *config, void *external)
```

#### Required Header

`stdmpi.h`

#### Description

**FilterConfigSet** sets a Filter’s *(filter)* configuration using data from the structure pointed to by *config*, and from the implementation-specific structure pointed to by *external* *(if external is not NULL)*.

The Filter’s configuration information in *external* is in addition to the Filter’s configuration information in *config*, i.e., the Filter’s configuration information in *config* and in *external* is not the same information. Note that *config* or *external* can be NULL (but not both NULL).

#### XMP Only

*external* either points to a structure of type MEIFilterConfig{} or is NULL.

#### Return Values

- **MPIMessageOK**
  - If *FilterConfigSet* successfully sets the Filter’s configuration using data from the structure(s)

#### See Also

- [mpiFilterConfigGet](https://support.motioneng.com/soft/filter/Method/cfset1.htm) | [MEIFilterConfig](https://support.motioneng.com/soft/filter/Method/cfset1.htm)
mpiFilterFlashConfigGet

Declaration

```c
long mpiFilterFlashConfigGet(MPIFilter filter, void *flash, MPIFilterConfig *config, void *external)
```

Required Header

`stdmpi.h`

Description

`FilterFlashConfigGet` gets a Filter’s (`filter`) flash configuration and writes it into the structure pointed to by `config`, and also writes it into the implementation-specific structure pointed to by `external` (if `external` is not NULL).

The Filter’s flash configuration information in `external` is in addition to the Filter’s flash configuration information in `config` and in `external` is not the same information. Note that `config` or `external` can be NULL (but not both NULL).

**XMP Only**

`external` either points to a structure of type `MEIFilterConfig`[] or is NULL.

Return Values

- `MPIMessageOK` if `FilterFlashConfigGet` successfully writes the Filter’s flash configuration to the structure(s)
  - `flash` is either an MEIFlash handle or MPIHandleVOID. If `flash` is MPIHandleVOID, an MEIFlash object will be created and deleted internally.

See Also

- `MEIFlash`
- `mpiFilterFlashConfigSet`
- `MEIFilterConfig`
**mpiFilterFlashConfigSet**

**Declaration**

```c
long mpiFilterFlashConfigSet(MPIFilter filter, void *flash,
                               MPIFilterConfig *config,
                               void *external)
```

**Required Header**

`stdmpi.h`

**Description**

`FilterFlashConfigSet` sets a Filter’s (`filter`) flash configuration using data from the structure pointed to by `config`, and also using data from the implementation-specific structure pointed to by `external` (if `external` is not NULL).

The Filter’s flash configuration information in `external` is in addition to the Filter’s flash configuration information in `config`, i.e., the flash configuration information in `config` and in `external` is not the same information. Note that `config` or `external` can be NULL (but not both NULL).

**XMP Only**

`external` either points to a structure of type `MEIFilterConfig` or is NULL.

**Return Values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPIMessageOK</td>
<td>If <code>FilterFlashConfigSet</code> successfully sets the Filter’s flash configuration using data from the structure(s) <code>flash</code> is either an MEIFlash handle or MPIHandleVOID. If <code>flash</code> is MPIHandleVOID, an MEIFlash object will be created and deleted internally.</td>
</tr>
</tbody>
</table>

**See Also**

- MEIFlash
- `mpiFilterFlashConfigGet`
- MEIFilterConfig
### mpiFilterGainGet

**Declaration**

```c
long mpiFilterGainGet (MPIFilter filter, long gainIndex, MPIFilterGain *gain)
```

**Required Header**

`stdmpi.h`

**Description**

`FilterGainGet` gets the gain coefficients of a Filter (``filter``) for the gain index specified by `gainIndex` and writes them into the structure pointed to by `gain`.

**Return Values**

- **MPIMessageOK**
  - if `FilterGainGet` successfully writes the gain coefficients to the structure

**See Also**

`mpiFilterGainSet`
### mpiFilterGainSet

**Declaration**

```c
long mpiFilterGainSet (MPIFilter filter, long gainIndex, MPIFilterGain *gain)
```

**Required Header**

```c
#include <stdmpi.h>
```

**Description**

`FilterGainSet` sets the gain coefficients of a Filter (`filter`, for the gain index specified by `gainIndex`) using data from the structure pointed to by `gain`.

**Return Values**

- `MPIMessageOK` if `FilterGainSet` successfully sets the gain coefficients of a Filter using data from the structure.

**See Also**

`mpiFilterGainGet`
**mpiFilterGainIndexGet**

**Declaration**

```c
long mpiFilterGainIndexGet(MPIFilter filter, long *gainIndex)
```

**Required Header**

`stdmpi.h`

**Description**

`FilterGainIndexGet` gets the current gain index of a Filter (`filter`) and writes it to the location pointed to by `gainIndex`.

**Return Values**

- `MPIMessageOK` if `FilterGainIndexGet` successfully writes the gain index to the location.

**See Also**

`mpiFilterGainIndexSet`
**mpiFilterGainIndexSet**

**Declaration**

```c
long mpiFilterGainIndexSet(MPIFilter filter, long gainIndex)
```

**Required Header**

stdmpi.h

**Description**

FilterGainIndexSet sets the current gain index of a Filter (filter) to gainIndex.

**Return Values**

- MPIMessageOK if FilterGainIndexSet successfully sets the current gain index to gainIndex

**See Also**

mpiFilterGainIndexGet
**mpiFilterStatus**

**Declaration**

```c
long mpiFilterStatus(MPIFilter filter, MPIStatus *status, void *external)
```

**Required Header**

`stdmpi.h`

**Description**

`FilterStatus` writes a Filter’s (`filter`) status into the structure pointed to by `status`, and also into the implementation-specific structure pointed to by `external` (if `external` is not NULL).

The Filter’s status information in `external` is in addition to the Filter’s status information in `status`, i.e., the status configuration information in `status` and in `external` is not the same information. Note that `status` or `external` can be NULL (but not both NULL).

**XMP Only**

`external` either points to a structure of type `MEIFilterStatus{}` or is NULL.

**Return Values**

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPIMessageOK</td>
<td>if <code>FilterStatus</code> successfully writes the status of a Filter to the structure(s)</td>
</tr>
</tbody>
</table>

**See Also**
mpiFilterMemory

Declaration  

```c
long mpiFilterMemory(MPIFilter filter, void **memory)
```

Required Header  
stdmpi.h

Description  

`FilterMemory` writes an address, which is used to access a Filter’s (`filter`) memory to the contents of `memory`. This address, or an address calculated from it, can be passed as the `src` parameter to `MPIFilterMemoryGet(...)` and as the `dst` parameter to `MPIFilterMemorySet(...)`. 

Return Values  

<table>
<thead>
<tr>
<th>Return Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPIMessageOK</td>
<td>if <code>FilterMemory</code> successfully writes the Filter’s memory address to the contents of <code>memory</code></td>
</tr>
</tbody>
</table>

See Also  

`mpiFilterMemoryGet` | `mpiFilterMemorySet`
mpiFilterMemoryGet

Declaration

```c
long mpiFilterMemoryGet (MPIFilter filter,
                        void      *dst,
                        void      *src,
                        long      count)
```

Required Header

```c
stdmpi.h
```

Description

`FilterMemoryGet` copies `count` bytes of a Filter’s (`filter`) memory (starting at address `src`) and writes them into application memory (starting at address `dst`).

Return Values

| MPIMessageOK       | if `FilterMemoryGet` successfully copies data from Filter memory to application memory |

See Also

`mpiFilterMemorySet` | `mpiFilterMemory`
mpiFilterMemorySet

Declaration
long mpiFilterMemorySet (MPIFilter filter,
void *dst,
void *src,
long count)

Required Header
stdmpi.h

Description
FilterMemorySet copies count bytes of application memory (starting at address src) and writes them into a Filter’s (filter) memory (starting at address dst).

Return Values

| MPIMessageOK | if FilterMemoryGet successfully copies data from application memory to Filter memory |

See Also
mpiFilterMemoryGet | mpiFilterMemory
**mpiFilterAxisMapGet**

**Declaration**

```c
long mpiFilterAxisMapGet(MPIFilter filter, MPIObjectMap *map)
```

**Required Header**

`stdmpi.h`

**Description**

`FilterAxisMapGet` gets the object map of the Axes that are associated with a Filter (`filter`), and writes it into the structure pointed to by `map`.

**Return Values**

| MPIMessageOK | if `FilterAxisMapGet` successfully writes the object map of Axes to the structure |

**See Also**

`mpiFilterAxisMapSet`
**mpiFilterAxisMapSet**

**Declaration**

```c
long mpiFilterAxisMapSet(MPIFilter filter, MPIObjectMap map)
```

**Required Header**

`stdmpi.h`

**Description**

*FilterAxisMapSet* sets the Axes associated with a Filter (*filter*), using data from the object map specified by *map*.

**Return Values**

- `MPIMessageOK` if `FilterAxisMapSet` successfully sets the Axes using the object map

**See Also**

`mpiFilterAxisMapGet`
mpiFilterControl

**Declaration**

```cpp
const MPIControl mpiFilterControl(MPIFilter filter)
```

**Required Header**

`stdmpi.h`

**Description**

`FilterControl` returns a handle to the motion controller (Control object) associated with the specified Filter object (`filter`).

**Return Values**

<table>
<thead>
<tr>
<th>handle</th>
<th>to a Control object that a Filter object is associated with</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPIHandleVOID</td>
<td>if the Filter object is invalid</td>
</tr>
</tbody>
</table>

**See Also**
**mpiFilterMotorMapGet**

**Declaration**

```c
long mpiFilterMotorMapGet (MPIFilter filter, MPIObjectMap *map)
```

**Required Header**

stdmpi.h

**Description**

`mpiFilterMotorMapGet` gets the object map of the Motors associated with the Filter (`filter`), and writes it into the structure pointed to by `map`.

**Return Values**

- **MPIMessageOK** if `mpiFilterMotorMapGet` successfully writes the object map of the Motors to the structure

**See Also**

`mpiFilterMotorMapSet`
**mpiFilterMotorMapSet**

**Declaration**

```c
long mpiFilterMotorMapSet(MPIFilter filter, MPIObjectMap map)
```

**Required Header**

`stdmpi.h`

**Description**

`FilterMotorMapSet` sets the Motors associated with the Filter (*filter*) using data from the object map specified by `map`.

**Return Values**

- **MPIMessageOK**
  
  if `FilterMotorMapGet` successfully sets the Motors using data from the object map

**See Also**

`mpiFilterMotorMapGet`
mpiFilterNumber

Declaration

long mpiFilterNumber(MPIFilter filter,
                        long *number)

Required Header

stdmpi.h

Description

For a motion controller that filter is associated with, FilterNumber writes the index of filter to the contents of number.

Return Values

MPIMessageOK if FilterNumber successfully writes the index of a Filter to the contents of number

See Also
**mpiFilterIntegratorReset**

**Declaration**

```c
long mpiFilterIntegratorReset(MPIFilter filter)
```

**Required Header**

`stdmpi.h`

**Description**

`FilterIntegratorReset` resets the integrators of `filter`.

**Return Values**

<table>
<thead>
<tr>
<th>MPIMessageOK</th>
<th>if <code>mpiFilterIntegratorReset</code> successfully clears the integrators of <code>filter</code>.</th>
</tr>
</thead>
<tbody>
<tr>
<td>MPIFilterMessageINVALID_ALGORITHM</td>
<td>if the <code>filter</code>'s current algorithm does not use integrators.</td>
</tr>
</tbody>
</table>

**Sample Code**

```c
/* Enable the amplifier for every motor attached to a motion supervisor */
void motionAmpEnable(MPIMotion motion)
{
    MPIControl control;
    MPIAxis axis;
    MPIMotor motor;
    MPIFilter filter;
    MPIObjectMap map;
    MPIObjectMap motionMotorMap;
    long motorIndex;
    long filterIndex;
    long returnValue;
    double position;
    long enableState;

    /* Get the controller handle */
    control = mpiMotionControl(motion);

    for (axis = mpiMotionAxisFirst(motion);
         axis != MPIHandleVOID;
         axis = mpiMotionAxisNext(motion, axis)) {
        /* Get the object map for the motors */
        returnValue = mpiAxisMotorMapGet(axis, &map);
        msgCHECK(returnValue);
        /* Add map to motionMotorMap */
        motionMotorMap |= map;
    }

    /* For every motor ... */
    for (motorIndex = 0; motorIndex < MEIXmpMAX_Motors; motorIndex++) {
        if (mpiObjectMapBitGET(motionMotorMap, motorIndex)) {
            /* Create motor handle */
            motor = mpiMotorCreate(control, motorIndex);
            msgCHECK(mpiMotorValidate(motor));

            /* Get the state of the amplifier */
            returnValue = mpiMotorAmpEnableGet(motor, &enableState);
            msgCHECK(returnValue);
        }
    }
}
```
/* If the amplifier is disabled ... */
if (enableState == FALSE) {

    /* For every axis */
    for (axis = mpiMotionAxisFirst(motion);
        axis != MPIHandleVOID;
        axis = mpiMotionAxisNext(motion, axis)) {

        /* Get the object map for the motors */
        returnValue = mpiAxisMotorMapGet(axis, &map);
        msgCHECK(returnValue);

        /* If axis is attached to motor ... */
        if (mpiObjectMapBitGET(map, motorIndex)) {

            /* Get the actual position of the axis */
            returnValue = mpiAxisActualPositionGet(axis, &position);
            msgCHECK(returnValue);

            /* Set command position equal to actual position */
            returnValue = mpiAxisCommandPositionSet(axis, position);
            msgCHECK(returnValue);

        }

    }

    /* Get the object map for the filters */
    returnValue = mpiMotorFilterMapGet(motor, &map);
    msgCHECK(returnValue);

    /* For every filter ... */
    for (filterIndex = 0;
        filterIndex < MEIXmpMAX_Filters;
        filterIndex++) {

        if (mpiObjectMapBitGET(map, filterIndex)) {

            /* Create filter handle */
            filter = mpiFilterCreate(control, filterIndex);
            msgCHECK(mpiFilterValidate(filter));

            /* Reset integrator */
            returnValue = mpiFilterIntegratorReset(filter);
            msgCHECK(returnValue);

            /* Delete filter handle */
            returnValue = mpiFilterDelete(filter);
            msgCHECK(returnValue);

        }

    }

    /* Enable the amplifier */
    returnValue = mpiMotorAmpEnableSet(motor, TRUE);
    msgCHECK(returnValue);

}

/* Delete motor handle */
returnValue = mpiMotorDelete(motor);
msgCHECK(returnValue);

}
See Also  
MPIFilter | MEIFilterConfig | MEIFilterGainPID | MEIFilterGainPIV  
mpiAxisActualPositionGet | mpiAxisCommandPositionSet

**Troubleshooting / Helpful Hints**

If an axis is not in an error state and the filter associated with that axis' motor has a non-zero integration term, then it is very likely that the integrator has built up a substantial integral term. Enabling the motor's amplifier when this has happened could cause the motor to jump with enormous force. Use `mpiFilterIntegratorReset` to reset the integrator before enabling the motor's amplifier to prevent this kind of jump.

Another condition that can cause the motor to jump upon enabling its amplifier is that the command position of the axis is not equal to the actual position of the axis. To prevent this situation, one should use `mpiAxisActualPositionGet` and `mpiAxisCommandPositionSet`. Please refer to these functions for a more in depth discussion.
MPIFilterCoeff

typedef union {
    float \textit{f};
    long \textit{l};
} \texttt{MPIFilterCoeff};

\textbf{Description}

| \textbf{f} | float coefficient |
| \textbf{l} | long coefficient  |

\textbf{See Also} \quad \texttt{MPIFilterCoeff\_COUNT\_MAX}
**MPIFilterConfig / MEIFilterConfig**

### MPIFilterConfig

```c
typedef struct MPIFilterConfig {
    long gainIndex;
    MPIFilterGain gain[MPIFilterGainCOUNT_MAX];
    long sercosGainIdnMap[MPIFilterCoeffCOUNT_MAX];
    MPIObjectMap axisMap;
    MPIObjectMap motorMap;
} MPIFilterConfig;
```

**Description**

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>gainIndex</td>
<td>Gain table index. Gain tables number 0 to MPIFilterGainCOUNT_MAX -1 (MPIFilterGainCOUNT_MAX = 4).</td>
</tr>
<tr>
<td>gain</td>
<td>See <a href="#">MPIFilterGain</a></td>
</tr>
<tr>
<td>sercosGainIdnMap</td>
<td>Map of Sercos servo loop gains. MPIFilterCoeffCOUNT_MAX coefficients in the map (MPIFilterCoeffCOUNT_MAX = 20).</td>
</tr>
<tr>
<td>axisMap</td>
<td>See <a href="#">MPIObjectMap</a></td>
</tr>
<tr>
<td>motorMap</td>
<td>See <a href="#">MPIObjectMap</a></td>
</tr>
</tbody>
</table>

### MEIFilterConfig

```c
typedef struct MEIFilterConfig {
    MEIXmpAlgorithm Algorithm;
    MEIXmpAxisInput Axis[MEIXmpFilterAxisInputs];
    long *VelPositionPtr;
    long *AuxInput[MEIXmpFilterAuxInputs];
    MEIXmpSwitchType GainSwitchType;
    float GainDelay;
    long GainWindow;
    MEIXmpSwitchType PPISwitchType;
    MEIXmpPPIMode PPIMode;
    float PPIDelay;
    long PPIWindow;
    MEIXmpIntResetConfig ResetIntegratorConfig;
    float ResetIntegratorDelay;
    MEIXmpPostFilter PostFilter;
} MEIFilterConfig;
```

See [MEIFilterConfig](http://support.motioneng.com/soft/filter/DataType/cf3.htm)
**Description**

<table>
<thead>
<tr>
<th><strong>Algorithm</strong></th>
<th>This value defines the algorithm that the filter is executing every servo cycle.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Axis[MEIXmpFilterAxisInputs]</strong></td>
<td>This array defines the axis (pointer to the axis) and coefficient for the position input into the filter. The input to the filter is the position error of the axis, which is multiplied by the coefficient defined by the Axis array.</td>
</tr>
<tr>
<td><strong>VelPositionPtr</strong></td>
<td>Velocity position pointer to an encoder input for algorithms that require a velocity encoder position input (such as the PIV algorithm).</td>
</tr>
<tr>
<td><strong>AuxInput[MEIXmpFilterAuxInputs]</strong></td>
<td>This array is a place holder for additional filter inputs from analog sources. This is currently not supported and is reserved for future use.</td>
</tr>
<tr>
<td><strong>GainSwitchType</strong></td>
<td>Value to define the gain table switch type. Not implemented in standard firmware.</td>
</tr>
<tr>
<td><strong>GainDelay</strong></td>
<td>Custom Delay not implemented in standard firmware.</td>
</tr>
<tr>
<td><strong>GainWindow</strong></td>
<td>Custom Delay not implemented in standard firmware.</td>
</tr>
<tr>
<td><strong>PPI SwitchType</strong></td>
<td>Value to define the gain switch type for PPI mode. Not implemented in standard firmware.</td>
</tr>
<tr>
<td><strong>PPIMode</strong></td>
<td>Value to define the PPI switch mode. Not implemented in standard firmware.</td>
</tr>
<tr>
<td><strong>PPI Delay</strong></td>
<td>Custom Delay not implemented in standard firmware.</td>
</tr>
<tr>
<td><strong>PPI Window</strong></td>
<td>Custom Window not implemented in standard firmware.</td>
</tr>
<tr>
<td><strong>Reset Integrator Config</strong></td>
<td>Value to define the integrator's reset configuration. Not supported in standard firmware.</td>
</tr>
<tr>
<td><strong>Reset Integrator Delay</strong></td>
<td>Value to define the integrator's reset delay. Not supported in standard firmware.</td>
</tr>
<tr>
<td><strong>PostFilter</strong></td>
<td>This array defines the configuration for the filter's post filter (the type, the length and values for the post filter coefficients).</td>
</tr>
</tbody>
</table>

**See Also**

MPIFilterConfig / MEIFilterConfig

MPIFilterGain

typedef struct MPIFilterGain {
    MPIFilterCoeff coeff[MPIFilterCoeffCOUNT_MAX];
} MPIFilterGain;

Description

| coeff | see MPIFilterCoeff |

See Also

MPIFilterGainCOUNT_MAX
**MEIFilterGainIndex**

```c
typedef enum { /* NOTE: These *must* match MEIXmpGain{}!!! */
    MEIFilterGainIndexINVALID,
    MEIFilterGainIndexSTOPPING2, /* MEIXmpGainSTOPPED2 */
    MEIFilterGainIndexSTOPPING1, /* MEIXmpGainSTOPPED1 */
    MEIFilterGainIndexSETTLING,  /* MEIXmpGainSETTLING */
    MEIFilterGainIndexMOVING,    /* MEIXmpGainMOVING */
    MEIFilterGainIndexLAST,      /* MEIXmpGainLAST */
    MEIFilterGainIndexDEFAULT   = MEIFilterGainIndexSTOPPING2,
} MEIFilterGainIndex;
```

**Description**

FilterGainIndex is a constant enum structure defining gain indexes used to select one of the four filter gain tables.

**See Also**
**MEIFilterGainPID**

**MEIFilterGainPID**

```c
typedef struct MEIFilterGainPID {
    struct {
        float proportional; /* Kp */
        float integral;    /* Ki */
        float derivative;  /* Kd */
    } gain;
    struct {
        float position;    /* Kpff */
        float velocity;   /* Kvff */
        float acceleration; /* Kaff */
        float friction;   /* Kfff */
    } feedForward;
    struct {
        float moving;       /* MovingIMax */
        float rest;         /* RestIMax */
    } integrationMax;
    long dRate;          /* DRate */
    struct {
        float limit;        /* OutputLimit */
        float limitHigh;    /* OutputLimitHigh */
        float limitLow;     /* OutputLimitLow */
        float offset;       /* OutputOffset */
    } output;
    struct {
        float positionFFT;  /* Ka0 */
        float filterFFT;    /* Ka1 */
        float velocityFFT;  /* Ka2 */
    } noise;
} MEIFilterGainPID;
```

**Description**

`FilterGainPID` is a structure that defines the filter coefficients for the PID filter algorithm.

**See Also**

- [High/Low Output Limits](#)
- [MEIFilterGainPIDCoeff](#)
MEIFilterGainPIDCoeff

typedef enum {
    MEIFilterGainPIDCoeffINVALID = -1,
    MEIFilterGainPIDCoeffGAIN_PROPORTIONAL,    /* Kp */
    MEIFilterGainPIDCoeffGAIN_INTEGRAL,         /* Ki */
    MEIFilterGainPIDCoeffGAIN_DERIVATIVE,       /* Kd */
    MEIFilterGainPIDCoeffFEEDFORWARD_POSITION,  /* Kpff */
    MEIFilterGainPIDCoeffFEEDFORWARD_VELOCITY,  /* Kvff */
    MEIFilterGainPIDCoeffFEEDFORWARD_ACCELERATION, /* Kaff */
    MEIFilterGainPIDCoeffFEEDFORWARD_FRICTION,  /* Kfff */
    MEIFilterGainPIDCoeffINTEGRATIONMAX_MOVING, /* MovingIMax */
    MEIFilterGainPIDCoeffINTEGRATIONMAX_REST,   /* RestIMax */
    MEIFilterGainPIDCoeffDRATE,                 /* DRate */
    MEIFilterGainPIDCoeffOUTPUT_LIMIT,          /* OutputLimit */
    MEIFilterGainPIDCoeffOUTPUT_LIMITHIGH,      /* OutputLimitHigh */
    MEIFilterGainPIDCoeffOUTPUT_LIMITLOW,       /* OutputLimitLow */
    MEIFilterGainPIDCoeffOUTPUT_OFFSET,         /* OutputOffset */
    MEIFilterGainPIDCoeffNOISE_POSITIONFFT,     /* Ka0 */
    MEIFilterGainPIDCoeffNOISE_FILTERFFT,       /* Ka1 */
    MEIFilterGainPIDCoeffNOISE_VELOCITYFFT,     /* Ka2 */
} MEIFilterGainPIDCoeff;

Description

FilterGainPIDCoeff is a structure of enums that defines the filter coefficients for the PID filter algorithm.

See Also

MEIFilterGainPID
MEIFilterGainPIV

typedef struct MEIFilterGainPIV {
    struct {
        float proportional; /* Kpp */
        float integral;     /* Kip */
    } gainPosition;
    struct {
        float proportional; /* Kpv */
    } gainVelocity1;
    struct {
        float position;      /* Kpff */
        float velocity;      /* Kvff */
        float acceleration;  /* Kaff */
        float friction;      /* Kfff */
    } feedForward;
    struct {
        float moving;        /* MovingIMax */
        float rest;          /* RestIMax */
    } integrationMax;
    struct {
        float feedback;      /* Kdv */
    } gainVelocity2;
    struct {
        float limit;         /* OutputLimit */
        float limitHigh;     /* OutputLimitHigh */
        float limitLow;      /* OutputLimitLow */
        float offset;        /* OutputOffset */
    } output;
    struct {
        float integral;      /* Kiv */
        float integrationMax; /* VintMax */
    } gainVelocity3;
    struct {
        float positionFFT;   /* Ka0 */
        float filterFFT;     /* Ka1 */
    } noise;
} MEIFilterGainPIV;

Description

FilterGainPIV is a structure that defines the filter coefficients for the PIV filter algorithm.

See Also

High/Low Output Limits section for special instructions regarding MEIFilterGainPIV.
MEIFilterGainPIVCoeff
**MEIFilterGainPIVCoeff**

typedef enum {
    MEIFilterGainPIVCoeffINVALID = -1,
    MEIFilterGainPIVCoeffGAINPOSITION_PROPORTIONAL, /* Kpp */
    MEIFilterGainPIVCoeffGAINPOSITION_INTEGRAL, /* Kip */
    MEIFilterGainPIVCoeffGAINVELOCITY_PROPORTIONAL, /* Kpv */
    MEIFilterGainPIVCoeffFEEDFORWARD_POSITION, /* Kpff */
    MEIFilterGainPIVCoeffFEEDFORWARD_VELOCITY, /* Kvff */
    MEIFilterGainPIVCoeffFEEDFORWARD_ACCELERATION, /* Kaff */
    MEIFilterGainPIVCoeffFEEDFORWARD_FRICTION, /* Kfff */
    MEIFilterGainPIVCoeffINTEGRATIONMAX_MOVING, /* MovingIMax */
    MEIFilterGainPIVCoeffINTEGRATIONMAX_REST, /* RestIMax */
    MEIFilterGainPIVCoeffGAINVELOCITY_FEEDBACK, /* Kdv */
    MEIFilterGainPIVCoeffOUTPUT_LIMIT, /* OutputLimit */
    MEIFilterGainPIVCoeffOUTPUT_LIMITHIGH, /* OutputLimitHigh */
    MEIFilterGainPIVCoeffOUTPUT_LIMITLOW, /* OutputLimitLow */
    MEIFilterGainPIVCoeffOUTPUT_OFFSET, /* OutputOffset */
    MEIFilterGainPIVCoeffGAINVELOCITY_INTEGRAL, /* Kiv */
    MEIFilterGainPIVCoeffGAINVELOCITY_INTEGRATIONMAX, /* Vintmax */
    MEIFilterGainPIVCoeffNOISE_POSITIONFFT, /* Ka0 */
    MEIFilterGainPIVCoeffNOISE_FILTERFFT, /* Kal */
} MEIFilterGainPIVCoeff;

**Description**

**FilterGainPIVCoeff** is a structure of enums that defines the filter coefficients for the PIV filter algorithm.

**See Also**

[High/Low Output Limits](http://support.motioneng.com/soft/filter/DataType/gnpivcoef2.htm) section for special instructions regarding MEIFilterGainPIV.

[MEIFilterGainPIV](http://support.motioneng.com/soft/filter/DataType/gnpivcoef2.htm)
**MEIFilterGainSercos**

**MEIFilterGainSercos**

typedef struct MEIFilterGainSercos {
  long        Coeff [MEIXmpSercosCoeffMAX];
  struct {
    long    velocity;
    long    acceleration;
    long    friction;
    float   scaler;
  } feedForward;
} MEIFilterGainSercos;

**Description**

FilterGainSercos is a structure that defines the filter coefficients for the SERCOS filter algorithm.

**See Also**

MEIFilterGainSercosCoeff
MEIFilterGainSercosCoeff

typedef enum {
    MEIFilterGainSercosCoeffINVALID = -1,
    MEIFilterGainSercosCoeff0,
    MEIFilterGainSercosCoeff1,
    MEIFilterGainSercosCoeff2,
    MEIFilterGainSercosCoeff3,
    MEIFilterGainSercosCoeff4,
    MEIFilterGainSercosCoeff5,
    MEIFilterGainSercosCoeff6,
    MEIFilterGainSercosCoeff7,
    MEIFilterGainSercosCoeff8,
    MEIFilterGainSercosCoeff9,
    MEIFilterGainSercosCoeff10,
    MEIFilterGainSercosCoeff11,
    MEIFilterGainSercosCoeff12,
    MEIFilterGainSercosCoeff13,
    MEIFilterGainSercosCoeff14,
    MEIFilterGainSercosCoeff15,
    MEIFilterGainSercosCoeffFEED_FORWARD_VELOCITY,
    MEIFilterGainSercosCoeffFEED_FORWARD_ACCELERATION,
    MEIFilterGainSercosCoeffFEED_FORWARD_FRICTION,
    MEIFilterGainSercosCoeffFEED_FORWARD_SCALER,
    MEIFilterGainSercosCoeffLAST,
    MEIFilterGainSercosCoeffFIRST = MEIFilterGainSercosCoeffINVALID + 1
} MEIFilterGainSercosCoeff;

Description

MEIFilterGainSercosCoeff is a structure of enums that defines the filter coefficients for the PID filter algorithm.

See Also

MEIFilterGainSercos
MPIFilterMessage

typedef enum {
    MPIFilterMessageFILTER_INVALID,  
    MPIFilterMessageINVALID_ALGORITHM,
} MPIFilterMessage;

Description

<table>
<thead>
<tr>
<th>Description</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>MPIFilterMessageFILTER_INVALID</td>
<td>This message indicates that an invalid filter number has been specified. This message is returned by the mpiFilterCreate method when the filter number is &lt; 0 or &gt; MEIXmpMAX_Filters</td>
</tr>
<tr>
<td>MPIFilterMessageINVALID_ALGORITHM</td>
<td>This message indicates that an invalid filter algorithm has been specified.</td>
</tr>
</tbody>
</table>

See Also

mpiFilterCreate
**MPIFilterCoeffCOUNT_MAX**

```c
#define MPIFilterCoeffCOUNT_MAX (20)
```

**Description**

`MPIFilterCoeffCOUNT_MAX` is a constant that defines the maximum number of filter coefficients contained in a gain table.

**See Also**

`MPIFilterCoeff`
**MPIFilterGainCOUNT_MAX**

MPIFilterGainCOUNT_MAX

```c
#define MPIFilterGainCOUNT_MAX (20)
```

**Description**

`FilterGainCOUNT_MAX` is a constant that defines the maximum number of filter gain tables. The first gain table is used by the standard filter types (all filter types except for the user filter type as defined by the structure MEIXmpAlgorithm). Additional gain tables can be used for manual or automatic gain switching. For firmware that implements automatic gain switching, please contact Motion Engineering. Manual gain switching can be accomplished by specifying the `gainIndex` of the `mpiFilterConfig` structure using the `mpiFilterConfigSet` method. Valid `gainIndex` values range from 0 to `MPIFilterGainCOUNT_MAX`.

**See Also**

MPIFilterGain
Special Note: *High / Low Output Limits (MEIFilterGainPID and PIV)*

In the 19990820 release, the [MEIFilterGainPID](http://support.motioneng.com/soft/filter/hi_lo_otpt.htm) and [MEIFilterGainPIV](http://support.motioneng.com/soft/filter/hi_lo_otpt.htm) structures were expanded to support High and Low output limits for PID and PIV algorithms. The "High" output limit prevents the filter output from exceeding the "High" value. The "Low" output limit prevents the filter output from falling below the "Low" value. This feature will allow an application to have upper and lower limits which are not centered on zero volts. If the "High" and "Low" values have the same sign, then the output will be limited to either the positive or negative range bounded by "High" and "Low".

The standard Output Limit is still valid. The controller will simultaneously use the standard Output Limit and the High / Low Output Limits to bound the output. The limits, (standard or high or low) that are closest to zero will be used as the boundary for the output.

Return to [MEIFilterGainPID](http://support.motioneng.com/soft/filter/hi_lo_otpt.htm) or [MEIFilterGainPIV](http://support.motioneng.com/soft/filter/hi_lo_otpt.htm)

Copyright © 2002
Motion Engineering