libmip Reference Manual

1.1

Generated by Doxygen 1.3.8

Wed Jun 1 09:19:07 2005

Contents

Chapter 1

libmip Hierarchical Index

1.1 libmip Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 2

libmip Class Index

2.1 libmip Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 3

libmip Class Documentation

3.1 EventClock Class Reference

Event clock.

`#include <EventClock.h>`

Public Types

- typedef unsigned long long **tTime**

  The datatype for time.

Public Member Functions

- **EventClock** ()

  Constructor.

- void **Update** ()

  Sets the time to now.

- unsigned long long **GetTime** ()

  Returns the clock in milliseconds.

- void **Report** (const std::string &name, std::ostream &out, int ccindent=0) const

  Writes a report of this object to the given stream. This is mainly useful for debugging.

3.1.1 Detailed Description

Event clock.

An object of this class is used to model the time reference for the application.

The time reference is updated by the **MainLoop**(p. 16) when new events (**EventInterface**(p. 7) timeout or active **IOInterface**(p. 11) filehandles) need to be dispatched. That means that

all events dispatched in that round perceive the same time (returned by the **GetTime()**(p. 5) method).

The documentation for this class was generated from the following file:

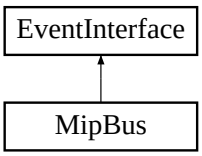
- EventClock.h

3.2 EventInterface Class Reference

General Event Interface.

`#include <EventInterface.h>`

Inheritance diagram for EventInterface::



Public Member Functions

- **EventInterface** (**EventInterfaceList** *evlist)

  Constructor.

- ~**EventInterface** ()

  Destructor.

- **EventInterfaceList** * **GetEventInterfaceList** ()

  Returns the **EventInterfaceList**(p. 9) *to which this EventInterface belongs to.*

- **EventInterface** * **GetEventInterfaceNext** ()

  Returns the next EventInterface in the list.

- void **DispatchEvent** ()

  Dispatches the event by calling **OnEvent()**(p. 7) *if necessary.*

- virtual void **OnEvent** ()=0

  This is called when the event timer expires.

- void **SetNextEvent** (int msec)

  Sets the next event (usec relative to the current event clock).

- void **SetNextEventAbsolute** (**EventClock::tTime** evtime)

  Sets the next event (absolute time).

- **EventClock::tTime GetNextEventTime** () const

  Returns the time of the next event.

- bool **HasNextEvent** () const

  Returns whether the event is active.

Protected Attributes

- **EventInterfaceList * mEventInterfaceList**

  The next EventInterface in the list.

- **EventInterface * mEventInterfaceNext**

  The next EventInterface in the list.

3.2.1 Detailed Description

General Event Interface.

This is an abstract class used to model an event timeout.

An class derived from EventInterface must implement the **OnEvent()**(p. 7) method. This method is called whenever the timeout occurs.

The most useful way to set a timeout is by calling the **SetNextEvent()**(p. 7) method. It takes as argument the number of milliseconds to wait until calling **OnEvent()**(p. 7). Note that the timeout is cleared when OnEvent is called, i.e. if you want OnEvent to be invoked every 100ms, you need to call

- SetNextEvent(100)

within the **OnEvent()**(p. 7) method to set the timeout again.

Alternatively, one may use the **SetNextEventAbsolute()**(p. 7) method to set an absolute time for the next event.

Note that an EventInterface only holds one timeout at the time. Calling **SetNextEvent()**(p. 7) or **SetNextEventAbsolute()**(p. 7) several times will only keep the smallest timeout (the event closest to now in time). However, you can create several objects derived from EventInterface and add them to the **EventInterfaceList**(p. 9).

The documentation for this class was generated from the following file:

- EventInterface.h

3.3 EventInterfaceList Class Reference

Implements a linked list for **EventInterface**(p. 7) objets.

`#include <EventInterfaceList.h>`

Public Member Functions

- **EventInterfaceList** ()

  Constructor.

- ~**EventInterfaceList** ()

  Destructor.

- **EventClock** * **GetEventClock** () const

  Returns the **EventClock**(p. 5).

- **EventInterface** * **GetEventInterfaceFirst** ()

  Returns the first **EventInterface**(p. 7) *object.*

- **EventInterface** * **GetEventInterfaceNext** (**EventInterface** *cur)

  Returns the next **EventInterface**(p. 7) *object.*

- **EventClock::tTime GetNextEventTimeout** ()

  Returns the timeout until the next event.

- void **UpdateClock** ()

  Updates the event clock.

- bool **AddEventInterface** (**EventInterface** *sif)

  Adds an **EventInterface**(p. 7) *to the list.*

- bool **RemoveEventInterface** (**EventInterface** *sif)

  Removes an **EventInterface**(p. 7) *from the list.*

- void **Report** (const std::string &name, std::ostream &out, int ccindent=0) const

  Writes a report of this object to the given stream. This is mainly useful for debugging.

Protected Attributes

- **EventInterface** * **mEventInterfaceFirst**

  The first **EventInterface**(p. 7) *object.*

- **EventClock** * **mEventClock**

  The event clock. It defines how the program experiences the flow of time.

3.3.1 Detailed Description

Implements a linked list for **EventInterface**(p. 7) objets.

The documentation for this class was generated from the following file:

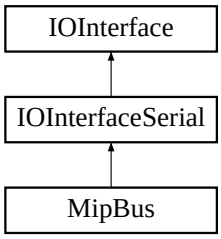
- EventInterfaceList.h

3.4 IOInterface Class Reference

General IO Interface.

`#include <IOInterface.h>`

Inheritance diagram for IOInterface::



Public Member Functions

- **IOInterface** (**IOInterfaceList** *iolist)
  Constructor.
- ~**IOInterface** ()
  Destructor.
- **IOInterfaceList** * **GetIOInterfaceList** ()
  Returns the **IOInterfaceList**(p. 13) *to which this IOInterface belongs to.*
- **IOInterface** * **GetIOInterfaceNext** ()
  Returns the next IOInterface in the list.
- virtual void **OnInputAvailable** ()
  (abstract) This is called by the application when the filehandle has become active.
- virtual void **OnDataArrival** (unsigned char *data, int len)=0
  (abstract) This is called by **OnInputAvailable()**(p. 11) *to process data that arrived.*
- virtual void **OnClose** ()=0
  (abstract) This is called when the filehandle is closed.
- virtual bool **Send** (unsigned char *data, int len)
  (abstract) Writes data.
- virtual bool **Send** (const std::string &str)
  (abstract) Writes a string.
- int **GetFileHandle** () const
  Returns the file handle.
- int **IsOpen** () const
  Returns whether the IO is open.

- bool **SetInputLogFile** (const std::string &file, bool append=true)

  Sets the input log file. A copy of all input is saved in this file. If the string is empty, no log is saved.

- bool **SetOutputLogFile** (const std::string &file, bool append=true)

  Sets the output log file. A copy of all output is saved in this file. If the string is empty, no log is saved.

Protected Attributes

- **IOInterfaceList** * **mIOInterfaceList**

  The next IOInterface in the list.

- **IOInterface** * **mIOInterfaceNext**

  The next IOInterface in the list.

- int **mFileHandle**

  The file handle of the open file/device (or -1 if the file/device is closed).

- std::ofstream **mLogInput**

  The file where incoming traffic is logged.

- std::ofstream **mLogOutput**

  The file where outgoing traffic is logged.

3.4.1 Detailed Description

General IO Interface.

An IOInterface object is attached to a file handle. The application should invoke OnInputAvailable whenever the filehandle becomes active. This class then reads the new data and dispatches it.

Data can be sent using the **Send()**(p. 11) method.

All incoming and outgoing data can be logged to a file using **SetInputLogFile()**(p. 12) and **SetOutputLogFile()**(p. 12).

This is an abstract class. You need to overwrite at least the **OnDataArrival()**(p. 11) and the **OnClose()**(p. 11) methods.

See also:
IOInterfaceSerial(p. 14), **IOInterfaceList**(p. 13)

The documentation for this class was generated from the following file:

- IOInterface.h

3.5 IOInterfaceList Class Reference

Implements a linked list for **IOInterface**(p. 11) objets.

`#include <IOInterfaceList.h>`

Public Member Functions

- **IOInterfaceList** ()
  Constructor.
- **IOInterface** * **GetIOInterfaceFirst** ()
  Returns the first **IOInterface**(p. 11) *object.*
- **IOInterface** * **GetIOInterfaceNext** (**IOInterface** *cur)
  Returns the next **IOInterface**(p. 11) *object.*
- bool **AddIOInterface** (**IOInterface** *sif)
  Adds an **IOInterface**(p. 11) *to the list.*
- bool **RemoveIOInterface** (**IOInterface** *sif)
  Removes an **IOInterface**(p. 11) *from the list.*
- void **Report** (const std::string &name, std::ostream &out, int ccindent=0) const
  Writes a report of this object to the given stream. This is mainly useful for debugging.

Protected Attributes

- **IOInterface** * **mIOInterfaceFirst**
  The first **IOInterface**(p. 11) *object.*

3.5.1 Detailed Description

Implements a linked list for **IOInterface**(p. 11) objets.

The documentation for this class was generated from the following file:

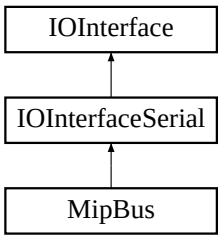
- IOInterfaceList.h

3.6 IOInterfaceSerial Class Reference

IOInterface(p. 11) for a serial connection.

`#include <IOInterfaceSerial.h>`

Inheritance diagram for IOInterfaceSerial::



Public Member Functions

- **IOInterfaceSerial** (**IOInterfaceList** *iolist, const std::string &device="", int baudrate=9600)

  Constructor.

- virtual void **OnBeforeClose** ()

  Called before a reset.

- virtual void **OnAfterOpen** ()

  Called after a reset.

- virtual void **OnAfterOpenError** ()

  Called after a reset.

- std::string **GetDevice** () const

  Returns the device.

- int **GetBaudRate** () const

  Returns the baud rate.

- bool **GetTermiosLocalMode** () const

  Returns the termios local mode flags.

- void **SetDevice** (const std::string &device)

  Sets the device. The new device is used after the next **Reopen()**(p. 15) *or* **Open()**(p. 15) *call.*

- void **SetBaudRate** (int baudrate)

  Sets the baud rate. The new baud rate is used after the next **Reopen()**(p. 15) *or* **Open()**(p. 15) *call.*

- void **SetTermiosLocalMode** (int mode)

  Sets the ICANON flag. The new setting is used after the next **Reopen()**(p. 15) *or* **Open()**(p. 15) *call.*

- bool **Open** ()

  Opens the serial port if it is closed. Returns true if the serial port was successfully opened or if it was open already. Note that calling **Open()**(p. 15) *on an open device won't do anything. Call* **Reopen()**(p. 15) *if you want to close and open it again.*

- bool **Close** ()

  Closes the serial port if it is open. Returns true if the serial port was successfully closed or if it was closed already.

- bool **Reopen** ()

  Closes (if necessary) and opens the serial port. Returns true if the serial port has been opened.

Protected Member Functions

- unsigned long **BaudRateConstant** (int baudrate)

  Converts a baud rate into a baud rate constant.

3.6.1 Detailed Description

IOInterface(p. 11) for a serial connection.

This implements an **IOInterface**(p. 11) for a serial port (usually /dev/ttyS∗).

The documentation for this class was generated from the following file:

- IOInterfaceSerial.h

3.7 MainLoop Class Reference

A typical main loop.

`#include <MainLoop.h>`

Public Member Functions

- **MainLoop** ()

  Constructor.

- ~**MainLoop** ()

  Destructor.

- int **Run** ()

  This executes the main loop until **Quit()**(p. 16) *is called.*

- void **Quit** ()

  Leaves the main loop.

- void **SetExitCode** (int code)

  Sets the exit code.

- int **GetExitCode** () const

  Returns the exit code.

- virtual void **OnInitialize** ()=0

  (abstract) This method is called right before entering the main loop.

- virtual void **OnTerminate** ()=0

  (abstract) This method is called right after leaving the main loop.

- virtual void **OnBeforeProcessing** ()

  (abstract) This method is called before the events are processed (after waiting for events).

- virtual void **OnAfterProcessing** ()

  (abstract) This method is called after the events are processed (before waiting for events).

- **IOInterfaceList** * **GetIOInterfaceList** () const

  Returns the **IOInterfaceList**(p. 13).

- **EventInterfaceList** * **GetEventInterfaceList** () const

  Returns the **EventInterfaceList**(p. 9).

- void **Report** (const std::string &name, std::ostream &out, int ccindent=0) const

  Writes a report of this object to the given stream. This is mainly useful for debugging.

Protected Attributes

- bool **mQuit**

  Set to true to leave the main loop.

- int **mExitCode**

  The exit code.

- **IOInterfaceList** * **mIOInterfaceList**

  The **IOInterface**(p. 11) *list.*

- **EventInterfaceList** * **mEventInterfaceList**

  The **EventInterface**(p. 7) *list.*

3.7.1 Detailed Description

A typical main loop.

This abstract class implements a main loop for a typical application. It listens to all handles of the attached **IOInterfaceList**(p. 13) and waits for all timeouts of the attached **EventInterfaceList**(p. 9).

To use this class, derive your main application class from it and overwrite the methods **OnInitialize()**(p. 16) and **OnTerminate()**(p. 16). Your application should first add all **EventInterface**(p. 7) to the **EventInterfaceList**(p. 9) and all IOInterfaces to the **IOInterfaceList**(p. 13). It may then call **Run()**(p. 16) to execute the main loop. Immediately before entering the main loop, **OnInitialize()**(p. 16) is called. Then, the program waits until

- one of the IOs becomes active (the corresponding **IOInterface**(p. 11) is invoked)
- a timeout occurs (the corresponding **EventInterface**(p. 7) is called)

Any such event may force the main loop to quit by calling **Quit()**(p. 16). The main loop then calls **OnTerminate()**(p. 16) and returns the exit code (set by SetExitCode).

The documentation for this class was generated from the following file:

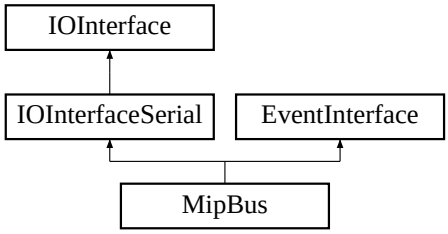
- MainLoop.h

3.8 MipBus Class Reference

Models a Mip bus.

`#include <MipBus.h>`

Inheritance diagram for MipBus::



Public Member Functions

- **MipBus** (**IOInterfaceList** *iolist, **EventInterfaceList** *evlist, const std::string &device, int baudrate=19200)

  Constructor.

- virtual void **OnDataArrival** (unsigned char *data, int len)

  (abstract) This is called by **OnInputAvailable()**(p. 11) *to process data that arrived.*

- virtual void **OnClose** ()

  (abstract) This is called when the filehandle is closed.

- virtual void **OnBeforeClose** ()

  Called before a reset.

- virtual void **OnAfterOpen** ()

  Called after a reset.

- virtual void **OnAfterOpenError** ()

  Called after a reset.

- void **OnEvent** ()

  This is called when the event timer expires.

- void **SetLogInterface** (**MipLogInterface** *miplog)

  Sets the mip log interface.

- **MipLogInterface** * **GetLogInterface** ()

  Returns the mip log interface.

- int **GetMipDeviceMax** ()

  Returns the maximum number of MipBus objects.

- **MipDevice** * **GetMipDevice** (int i)

Returns a Mip device.

- bool **AddMipDevice** (**MipDevice** *mip)

  Adds a Mip device.

- bool **RemoveMipDevice** (**MipDevice** *mip)

  Removes a Mip device.

- bool **IsIdle** ()

  Returns whether the bus is idle (no pending command).

- **MipCommand** * **GetCurrentCommand** ()

  Returns the current command (or 0 if the bus is idle).

- void **SendNextCommand** ()

  Sends the next command in the queue (if the bus is free and the queue is not empty).

- void **CancelCommand** ()

  Aborts the current command.

- void **CancelAllCommands** ()

  Aborts the current command and flushes the command queue.

- void **CheckOnline** ()

  Checks the online status and opens or closes the serial port if necessary.

- void **Report** (const std::string &name, std::ostream &out, int ccindent=0) const

  Writes a report of this object to the given stream. This is mainly useful for debugging.

Protected Member Functions

- bool **GetNextCommand** (int from, int to)

  Asks a set of **MipDevice**(p. 23) *objects for a new command and sets it as the current command if found.*

- void **SendPacket** (unsigned char *packet, int len) const

  Sends a packet.

Protected Attributes

- **MipDevice** * **mMipDevice** [**mMipDeviceMax**]

  Mip devices on the bus.

- **MipCommand** * **mCurrentMipCommand**

  The current Mip command.

- int **mTimeout**

Timeout in milliseconds.

- int **mNextMipDeviceIndex**

  The next mip device to ask for a mip comand.

- **MipLogInterface** * **mMipLogInterface**

  The log interface.

Static Protected Attributes

- const int **mMipDeviceMax** = 128

  Maximum number of **MipDevice**(p. 23) *objects.*

3.8.1 Detailed Description

Models a Mip bus.

A Mip bus contains several **MipDevice**(p. 23) objects with are identified by their device ID.

Due to protocol restrictions (to avoid collisions), only one command can be sent over the same serial port (MIP bus) at a time. The next command can be sent only after a reply for the current command has been received. Therefore, commands are queued in the MipDevices. If several MipDevices are attached to the same MipBus, they are served in a round robin fashion.

The queue length is currently limited to 16 commands. Further MipCommands are silently discarded.

The documentation for this class was generated from the following file:

- MipBus.h

3.9 MipCommand Class Reference

MipBus(p. 18) Protocol Command.

`#include <MipCommand.h>`

Public Types

- enum **eStatus** {
  sNone, **sQueued**, **sNotSent**, **sSent**,
  sAckReceived, **sReceived**, **sErronous**, **sCancelled** }
  Status flags.

Public Member Functions

- **MipCommand** (int cmdid, **MipTxMessage** *txmsg)
  Constructor.
- ~**MipCommand** ()
  Destructor.
- int **GetID** () const
  Returns the command ID.
- **MipRxMessage** * **GetRxMessage** () const
  Returns the message to send.
- **MipTxMessage** * **GetTxMessage** () const
  Returns the received message.
- **MipDevice** * **GetMipDevice** () const
  Returns the **MipBus**(p. 18).
- **eStatus GetStatus** () const
  Returns the status of the command.
- unsigned char **GetAck** () const
  Returns the status of the command.
- void **SetCommandID** (int id)
  Sets the command ID.
- void **SetRxMessage** (**MipRxMessage** *msg)
  Sets the message to send.
- void **SetTxMessage** (**MipTxMessage** *msg)
  Sets the received message.

- void **SetMipDevice** (**MipDevice** *mipdevice)

  Sets the **MipBus**(p. 18).

- void **SetStatus** (**eStatus** status)

  Sets the status of the command.

- void **SetAck** (unsigned char ack)

  Sets the ack of the command.

3.9.1 Detailed Description

MipBus(p. 18) Protocol Command.

The MipCommand class is the abstraction used when sending a command to a MIP device. It mainly consists of a **MipTxMessage**(p. 38) (the message transmitted to the MIP) and a **MipRx-Message**(p. 36) (the reply received from the MIP). Furthermore, it contains a comman ID which not transmitted to the MIP.

A MIP command may be in various states (eStatus). The most important states are:

- sQueued: The command is in the queue, waiting for being transmitted.
- sNotSent: The command has been dequeued, but sending has not completed yet.
- sSent: The command has been sent but no reply has been received yet.
- sAckReceived: An ack was received but the complete reply is still awaited.
- sReceived: A complete and correct reply has been received.
- sErronous: An erroneous reply has been received (wrong CRC).
- sCancelled: The command has been cancelled.

The documentation for this class was generated from the following file:

- MipCommand.h

3.10 MipDevice Class Reference

Models a Mip device on a Mip bus.

`#include <MipDevice.h>`

Public Member Functions

- **MipDevice** (**MipBus** *mipbus=0, int deviceid=0)

  Constructor.

- **MipDevice** (**MipManager** *mm, const std::string &device, int baudrate, int deviceid)

  Constructor.

- **MipBus** * **GetMipBus** () const

  Returns the **MipBus**(p. 18) *object.*

- int **GetDeviceID** () const

  Returns the device id (slave).

- bool **Open** ()

  Opens the MipDevice. The corresponding **MipBus**(p. 18) *opens the serial interface automatically if necessary.*

- bool **Close** ()

  Closes the MipDevice. The corresponding **MipBus**(p. 18) *closes the serial interface automatically if no other MipDevice is open on this* **MipBus**(p. 18).

- bool **IsOpen** ()

  Returns whether the MipDevice is open.

- void **OnAckReceived** (**MipCommand** *cmd)

  Called when an ACK is received. The MIP sends an ACK after having received the message and checked its checksum.

- void **OnPacketReceived** (**MipCommand** *cmd)

  Called when the response packet has been received successfully.

- void **OnErronousPacketReceived** (**MipCommand** *cmd)

  Called when the response packet has been received erronously (wrong packet checksum, wrong length checksum).

- bool **AddCommand** (**MipCommand** *cmd)

  Adds a command to the queue.

- bool **AddPriorityCommand** (**MipCommand** *cmd)

  Adds a command to the beginning of the queue. It will be executed just after the current.

- void **FlushCommandQueue** ()

  Deletes the command queue. This does not abort the current command.

- **MipCommand** * **GetNextCommand** ()

  Switches to the next command.

- int **GetMipCommandMax** ()

  Returns the maximum number of commands.

- **MipCommand** * **GetMipCommand** (int i)

  Returns a **MipCommand**(p. 21).

- void **SetMipBus** (**MipBus** *MipBus)

  Sets the **MipBus**(p. 18) *object.*

- void **SetDeviceID** (int deviceid)

  Sets the device ID (slave).

- void **SetDevice** (**MipManager** *mm, const std::string &device, int baudrate, int deviceid)

  Sets the mip bus and the device id. The mip bus is determined using the **MipManager**(p. 31) *object.*

- void **SetDevice** (**MipBus** *mipbus, int deviceid)

  Sets the mip bus and the device id.

- bool **SendCommand** (int cmdid, **MipMessage::eType** type)
- bool **SendCommand_SignedInteger32** (int cmdid, **MipMessage::eType** type, int param1)
- bool **SendCommand_Float** (int cmdid, **MipMessage::eType** type, float param1)
- bool **SendCommand_Byte** (int cmdid, **MipMessage::eType** type, unsigned char param1)
- bool **SendCommand_ByteFloatFloat** (int cmdid, **MipMessage::eType** type, unsigned char param1, float param2, float param3)
- bool **SendReadAxisStatus** (int cmdid=0)
- bool **SendReadError** (int cmdid=0)
- bool **SendClearError** (int cmdid=0)
- bool **SendReadWarning** (int cmdid=0)
- bool **SendClearWarning** (int cmdid=0)
- bool **SendReadVersion** (int cmdid=0)
- bool **SendSetRegMode** (int cmdid, unsigned char mode)
- bool **SendSetPosVelocity** (int cmdid, float velocity)
- bool **SendSetWaitMode** (int cmdid, unsigned char mode)
- bool **SendMoveAbsolute** (int cmdid, int position)
- bool **SendMoveRelative** (int cmdid, int relposition)
- bool **SendStopMotion** (int cmdid=0, unsigned char stopmode=0)
- bool **SendFindHomeSys** (int cmdid=0, int timeoutms=0)
- bool **SendReadPosition** (int cmdid=0)
- bool **SendReadVelocity** (int cmdid=0)
- bool **SendReadVelocityMean** (int cmdid=0)
- bool **SendReadCurrent** (int cmdid=0)
- bool **SendCheckCRC** (int cmdid, unsigned char selection)
- virtual void **OnReceivedReadAxisStatus** (int cmdid, **eAxisStatusFlags** status)=0
- virtual void **OnReceivedReadError** (int cmdid, **eError** error)=0

- virtual void **OnReceivedClearError** (int cmdid)=0
- virtual void **OnReceivedReadWarning** (int cmdid, **eWarning** warning)=0
- virtual void **OnReceivedClearWarning** (int cmdid)=0
- virtual void **OnReceivedReadVersion** (int cmdid, int command_interpreter_version, int runtime_library_version, int hardware_revision, int bootrom_version, int application_number, int application_version)=0
- virtual void **OnReceivedSetPosVelocity** (int cmdid)=0
- virtual void **OnReceivedSetWaitMode** (int cmdid)=0
- virtual void **OnReceivedMoveAbsolute** (int cmdid)=0
- virtual void **OnReceivedMoveRelative** (int cmdid)=0
- virtual void **OnReceivedStopMotion** (int cmdid)=0
- virtual void **OnReceivedFindHomeSys** (int cmdid)=0
- virtual void **OnReceivedReadPosition** (int cmdid, int value)=0
- virtual void **OnReceivedReadVelocity** (int cmdid, int value)=0
- virtual void **OnReceivedReadVelocityMean** (int cmdid, int value)=0
- virtual void **OnReceivedReadCurrent** (int cmdid, int value)=0
- virtual void **OnReceivedCheckCRC** (int cmdid, bool ok, int crc)=0
- void **Report** (const std::string &name, std::ostream &out, int ccindent=0) const

  Writes a report of this object to the given stream. This is mainly useful for debugging.

Protected Types

- enum **eError** {

  sErrorNone = 0, **sErrorCurofs0** = 1, **sErrorCurofs1** = 2, **sErrorHall** = 3,

  sErrorGatedrv = 4, **sErrorNobrake** = 5, **sErrorHwconfig** = 7, **sErrorEncoder** = 6,

  sErrorEncodersw = 8, **sErrorDigovld** = 10, **sErrorUplimit** = 11, **sErrorLolimit** = 12,

  sErrorPosreg = 20, **sErrorLeft** = 21, **sErrorRight** = 22, **sErrorHome** = 23,

  sErrorIndex = 25, **sErrorAmpdisab** = 24, **sErrorStop** = 40, **sErrorHalt** = 42,

  sErrorBrake = 43, **sErrorCurmax** = 41, **sErrorOvrcur** = 44, **sErrorOvrtemp** = 45,

  sErrorComwatch = 46, **sErrorWatchdog** = 47, **sErrorBatch** = 48 }

  The MIP error codes.

- enum **eWarning** {

  sWarnNone = 0, **sWarnUnknown** = 100, **sWarnSyntax** = 110, **sWarnLine** = 120,

  sWarnRange = 130, **sWarnDef** = 140, **sWarnError** = 150, **sWarnForbid** = 160,

  sWarnNomove = 170, **sWarnUplimit** = 171, **sWarnLolimit** = 172, **sWarnUnreachable** = 173,

  sWarnTimeout = 174, **sWarnPosregoff** = 180, **sWarnCurregoff** = 181, **sWarnAmpdisab** = 182,

  sWarnStop = 183, **sWarnLeft** = 184, **sWarnRight** = 185, **sWarnReset** = 186,

  sWarnHardwareReset = 187, **sWarnFlash** = 200 }

  The MIP warning codes.

- enum **eAxisStatusFlags** {

  sAxisStatusMoving = 0x1, **sAxisStatusCteVel** = 0x2, **sAxisStatusSinus** = 0x4, **sAxisStatusVelMode** = 0x8,

  sAxisStatusCmdActive = 0x10, **sAxisStatusWaitEnd** = 0x20, **sAxisStatusLoLimit** = 0x40, **sAxisStatusHiLimit** = 0x80,

  sAxisStatusError = 0x100, **sAxisStatusWarning** = 0x200, **sAxisStatusCurReg** = 0x400, **sAxisStatusPosReg** = 0x800,

  sAxisStatusReferenced = 0x1000, **sAxisStatusHome** = 0x2000, **sAxisStatusRight** = 0x4000, **sAxisStatusLeft** = 0x8000 }

  The axis status flags as defined in the MIP documentation. These flags are returned by the ReadAxisStatus command.

- enum **eTrajectoryFlags** {

  sTrajectoryDone = 0x1, **sTrajectoryOvershoot** = 0x2, **sTrajectoryTrapez** = 0x4, **sTrajectoryVmaxNeg** = 0x8,

  sTrajectoryUpLimit = 0x10, **sTrajectoryLoLimit** = 0x20, **sTrajectoryForbidden** = 0x40 }

  The trajectory flags as defined in the MIP documentation.

3.10.1 Detailed Description

Models a Mip device on a Mip bus.

A MipDevice objects represents one physical MIP. A MIP is addressed with a device ID on a serial bus. The serial bus is modelled with a **MipBus**(p. 18) object and the **MipManager**(p. 31) handles a list of **MipBus**(p. 18) objects.

This class should not be used directly. It is intended to be inherited by a subclass implementing the desired functionality. MipDevice objects can then be created as follows:

- **MipManager**(p. 31) *mm=new MipManager();
- MyMip *md1=MyMip(mm, "/dev/ttyS0", 4800, 1)
- MyMip *md2=MyMip(mm, "/dev/ttyS0", 4800, 2) This creates two MyMip objects (which inherit from the MipDevice class) on the same serial interface but with a different device ID. The **MipManager**(p. 31) automatically creates a **MipBus**(p. 18) object for the serial interface "/dev/ttyS0" and attaches the two MyMip objects with their respective device IDs.

3.10.2 Member Enumeration Documentation

3.10.2.1 enum MipDevice::eError [protected]

The MIP error codes.

Enumeration values:

sErrorNone not in error state

sErrorCurofs0 Bad offset in current measurement MAX.

sErrorCurofs1 Bad offset in current measurement MAX.

sErrorHall Bad hall signals combination MAX.

sErrorGatedrv Gate driver : Undervoltage MAX.
sErrorNobrake Braking failed (motor +/- reversal ?) MAX.
sErrorHwconfig Hardware configuration error MAX.
sErrorEncoder Bad encoder signal MID.
sErrorEncodersw missing encoder signals (SW detection) MID
sErrorDigovld Digital output overload STD.
sErrorUplimit Target position out of upper bond STD.
sErrorLolimit Target position out of lower bond STD.
sErrorPosreg Position error too large STD.
sErrorLeft Left switch encountered STD.
sErrorRight Right switch encountered STD.
sErrorHome Home switch not found STD.
sErrorIndex Index not found STD.
sErrorAmpdisab Amplifier has been disabled STD.
sErrorStop Stop was activated MID.
sErrorHalt Regulation stopped by 'Halt' command MID.
sErrorBrake Motor braked by 'Brake' command STD.
sErrorCurmax Motor current average to high STD.
sErrorOvrcur Overcurrent on power stage STD.
sErrorOvrtemp Overtemperature on power stage STD.
sErrorComwatch Communication watchdog STD.
sErrorWatchdog System watchdog (Firmware failure) MAX.
sErrorBatch Batch execution aborted STD.

3.10.2.2 enum MipDevice::eWarning [protected]

The MIP warning codes.

Enumeration values:
sWarnNone No warning.
sWarnUnknown Unknown command.
sWarnSyntax Syntax error.
sWarnLine Command line too long.
sWarnRange Value out of range.
sWarnDef Parameters not fully defined.
sWarnError Error not cleared.
sWarnForbid Command not allowed at the moment.
sWarnNomove Motion not started.
sWarnUplimit Target position out of bonds.
sWarnLolimit Target position out of bonds.
sWarnUnreachable Position cannot be reached.
sWarnTimeout Requested action took to much time.

sWarnPosregoff Pos. Regulation is switched off.

sWarnCurregoff Cur. Regulation is switched off.

sWarnAmpdisab Power stage is disabled.

sWarnStop STOP still activated.

sWarnLeft LEFT still activated.

sWarnRight RIGHT still activated.

sWarnReset RESET is required.

sWarnHardwareReset hardware RESET is required

sWarnFlash Problem with flash.

The documentation for this class was generated from the following file:

- MipDevice.h

3.11 MipLogInterface Class Reference

Receiver of log messages from the **MipManager**(p. 31) object and the attached **MipBus**(p. 18) objects.

`#include <MipLogInterface.h>`

Public Member Functions

- **MipLogInterface** (**MipManager** *mipmanager)

  Constructor.

- void **SetMipManager** (**MipManager** *mipmanager)

  Sets the **MipManager**(p. 31). *This method is automatically invoked when mipmanager->SetLogInterface is called.*

- **MipManager** * **GetMipManager** ()

  Returns the **MipManager**(p. 31).

- virtual void **OnDataReceived** (**MipBus** *mipbus, unsigned char *data, int len)=0

  (abstract) Called when data is received.

- virtual void **OnUnexpectedDataReceived** (**MipBus** *mipbus, unsigned char *data, int len)=0

  (abstract) Called when unexpected data is received. That means, some data is received without having issued a command. This may happen if another device is on the same bus or if a Mip is executing some command when it is plugged in.

- virtual void **OnSendPacket** (**MipBus** *mipbus, const unsigned char *data, int len)=0

  (abstract) Called before a packet is sent.

- virtual void **OnBeforeClose** (**MipBus** *mipbus)=0

  (abstract) Called before the serial port of a **MipBus**(p. 18) *is closed.*

- virtual void **OnAfterOpen** (**MipBus** *mipbus)=0

  (abstract) Called right after opening the serial port of a **MipBus**(p. 18).

- virtual void **OnAfterOpenError** (**MipBus** *mipbus)=0

  (abstract) Called if an error occured when opening a serial port.

Protected Member Functions

- std::string **DataToString** (unsigned char *packet, int len)

  Turns a piece of data (e.g. a packet) into hexadecimal representation.

Protected Attributes

- **MipManager** * **mMipManager**

  The associated **MipManager**(p. 31) *object.*

3.11.1 Detailed Description

Receiver of log messages from the **MipManager**(p. 31) object and the attached **MipBus**(p. 18) objects.

This is an abstract interface intended for logging. It allows to catch various internal events of the **MipManager**(p. 31) and the **MipBus**(p. 18) objects.

To use this interface, create a subclass of it and implement its abstract methods. Attach your class to the **MipManager**(p. 31) using:

- mymipmanager->SetLogInterface(mymiplogclass); It is recommended to do this immediately after creating the **MipManager**(p. 31) object.

The documentation for this class was generated from the following file:

- MipLogInterface.h

3.12 MipManager Class Reference

Manages a Set of **MipBus**(p. 18) objects with attached MipDevices.

`#include <MipManager.h>`

Public Member Functions

- **MipManager** (**IOInterfaceList** *iolist=0, **EventInterfaceList** *evlist=0)
  Constructor.
- ~**MipManager** ()
  Destructor.
- void **SetLogInterface** (**MipLogInterface** *miplog)
  Sets the mip log interface.
- **MipLogInterface** * **GetLogInterface** ()
  Returns the mip log interface.
- **IOInterfaceList** * **GetIOInterfaceList** ()
  Returns the IO interface list.
- **EventInterfaceList** * **GetEventInterfaceList** ()
  Returns the event interface list.
- **MipBus** * **FindMipBus** (const std::string &device)
  Returns a **MipBus**(p. 18) *object.*
- int **GetMipBusMax** ()
  Returns the maximum number of **MipBus**(p. 18) *objects.*
- **MipBus** * **GetMipBus** (int i)
  Returns a **MipBus**(p. 18) *object.*
- void **Report** (const std::string &name, std::ostream &out, int ccindent=0) const
  Writes a report of this object to the given stream. This is mainly useful for debugging.

3.12.1 Detailed Description

Manages a Set of **MipBus**(p. 18) objects with attached MipDevices.

This is the first object to create when working with this library. All **MipDevice**(p. 23) objects are attached to it.

The MipManager requires an **IOInterfaceList**(p. 13) and an **EventInterfaceList**(p. 9). If these arguments are not provided, the MipManager creates these lists for you and you can access them using GetIOInterfaceList and GetEventInterfaceList. Note that if you don't use the **MainLoop**(p. 16) implementation of this library as your program's main loop, you need to listen to all

filehandles of the **IOInterfaceList**(p. 13) and call the OnInputAvailable() method on the corresponding **IOInterface**(p. 11) object when filehandles become active.

The MipManager transparently maintains a list of **MipBus**(p. 18) objects. If you add a **MipDevice**(p. 23) on a serial port, a **MipBus**(p. 18) object for that serial port is created (unless such an object exists already) and the **MipDevice**(p. 23) object is attached to this **MipBus**(p. 18). Hence, the structure looks like this:

- MipManager
  - **MipBus**(p. 18) /dev/ttyS0
    - **MipDevice**(p. 23) ID=1
    - **MipDevice**(p. 23) ID=2
    - ...
  - **MipBus**(p. 18) /dev/ttyS1
    - **MipDevice**(p. 23) ID=1
    - ...
  - **MipBus**(p. 18) ...

You can use the **Report()**(p. 31) method to dump the data structure of the MipManager.

The documentation for this class was generated from the following file:

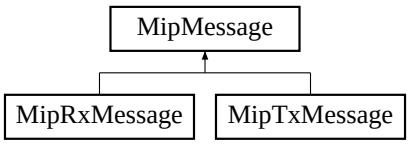
- MipManager.h

3.13 MipMessage Class Reference

MipBus(p. 18) Protocol Message.

`#include <MipMessage.h>`

Inheritance diagram for MipMessage::



Public Types

- enum **eType** {

  sNone = 0x00, **sReset** = 0x08, **sHalt** = 0x09, **sBrake** = 0x0A,

  sReadAxisStatus = 0x0B, **sReadError** = 0x0C, **sClearError** = 0x0D, **sReadWarning** = 0x0E,

  sClearWarning = 0x0F, **sReadTempParam** = 0x10, **sReadPermanentParam** = 0x11, **sSetTempParam** = 0x12,

  sResetTempParam = 0x14, **sUpdatePermParam** = 0x15, **sReadAllTempParam** = 0x16, **sReadAllPermParam** = 0x17,

  sSetAllTempParam = 0x18, **sSetAllPermParam** = 0x19, **sReadCurrentGains** = 0x1A, **sSetCurrentGains** = 0x1B,

  sReadPositionGains = 0x1C, **sSetPositionGains** = 0x1D, **sReadVersion** = 0x1F, **sSetRegMode** = 0x20,

  sSetProfile = 0x21, **sSetPosVelocity** = 0x22, **sSetWaitMode** = 0x23, **sMoveAbsolute** = 0x24,

  sMoveRelative = 0x25, **sWaitForTargetPos** = 0x26, **sSetVelocity** = 0x27, **sStopMotion** = 0x28,

  sFindHomeSys = 0x29, **sFindHome** = 0x2A, **sFindIndex** = 0x2B, **sDefinePosition** = 0x2C,

  sWaitForPosition = 0x2D, **sReadIndexDistance** = 0x2E, **sReadProfile** = 0x2F, **sReadPosition** = 0x30,

  sReadPositionMust = 0x31, **sReadPositionIsMust** = 0x32, **sReadVelocity** = 0x33, **sReadVelocityMean** = 0x34,

  sReadVelocityMust = 0x35, **sReadVelocityIsMust** = 0x36, **sReadCurrent** = 0x37, **sReadCurrentMean** = 0x38,

  sReadCurrentMust = 0x39, **sReadCurrentIsMust** = 0x3A, **sReadPosVelCur** = 0x3B, **sReadPosVelCurMean** = 0x3C,

  sReadPosVelCurMust = 0x3D, **sReadPosVelCurIsMust** = 0x3E, **sReadAxisInfo** = 0x3F, **sReadInput** = 0x40,

  sReadInputByte = 0x41, **sWaitForInput** = 0x42, **sSetOutput** = 0x43, **sSetOutputByte** = 0x44,

  sReadAnalogInput = 0x45, **sSetUserPWM** = 0x46, **sReadSysTime** = 0x47, **sWaitTime** = 0x48,

sSetCurrent = 0x49, **sReadECStatus** = 0x4A, **sReadMaxStack** = 0x4F, **sPutString** = 0x4B,

sGetString = 0x4C, **sPutNumber** = 0x4D, **sGetNumber** = 0x4E, **sInitMoveAbsolute** = 0x50,

sInitMoveRelative = 0x51, **sInitSetVelocity** = 0x52, **sStartMovement** = 0x53, **sInitAutoTune** = 0x60,

sAutotune = 0x61, **sSquareCurrentTest** = 0x62, **sSetupRecorder** = 0x68, **sRecordData** = 0x69,

sReadNbOfSamples = 0x6A, **sReadRecordedData** = 0x6B, **sReadAllTempUsr1Param** = 0x70, **sReadAllPermUsr1Param** = 0x71,

sSetAllTempUsr1Param = 0x72, **sSetAllPermUsr1Param** = 0x73, **sReadAllTempUsr2Param** = 0x74, **sReadAllPermUsr2Param** = 0x75,

sSetAllTempUsr2Param = 0x76, **sSetAllPermUsr2Param** = 0x77, **sCheckCRC** = 0x78 }

Defines the enum eType. The MIP message types.

Public Member Functions

- **MipMessage** (int src=0, int dest=0, **eType** type=sNone)

  Constructor.

- int **GetDestination** () const

  Returns the destination.

- int **GetSource** () const

  Returns the source.

- **eType GetType** () const

  Returns the type.

- virtual int **GetLength** () const =0

  Returns the length.

- void **SetDestination** (int dest)

  Sets the destination.

- void **SetBroadcast** ()

  Sets the broadcast destination.

- void **SetSource** (int src)

  Sets the source.

- void **SetType** (**eType** type)

  Sets the message type.

- unsigned int **CalcCRC** ()

  Calculates the checksum of the packet.

Protected Attributes

- int **mDestination**

  The device ID of the destination (the device that receives the message).

- int **mSource**

  The device ID of the source (the device that sends the message).

- **eType mType**

  The message type.

- unsigned char **mData** [1024]

  The data.

3.13.1 Detailed Description

MipBus(p. 18) Protocol Message.

This (abstract) class models a MIP bus message. MIP bus messages are implemented in the classes **MipRxMessage**(p. 36) (received messages) and **MipTxMessage**(p. 38) (messages to transmit).

For information about the MIP bus message format, please refer to the MIP manual.

The documentation for this class was generated from the following file:

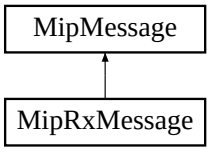
- MipMessage.h

3.14 MipRxMessage Class Reference

MipBus(p. 18) Protocol Receive Message.

`#include <MipRxMessage.h>`

Inheritance diagram for MipRxMessage::



Public Member Functions

- **MipRxMessage** ()

  Constructor.

- int **GetLength** () const

  Returns the length.

- bool **IsErronous** () const

  Returns whether the packet is erronous or not. (Erronous means that either the length checksum or the packet checksum are wrong.) An erronous packet is always marked as complete.

- bool **IsComplete** () const

  Returns whether the packet is complete or not. (Not complete means that not all of it has been received yet.).

- void **AddReceivedData** (unsigned char *data, int len)

  Adds received data.

- unsigned char **ReadCommand** ()

  Reads a command (byte) from the data.

- unsigned char **ReadByte** ()

  Reads a byte from the data.

- unsigned short int **ReadUnsignedInteger16** ()

  Reads an unsigned 16bit integer from the data.

- short int **ReadSignedInteger16** ()

  Reads a signed 16bit integer from the data.

- unsigned int **ReadUnsignedInteger32** ()

  Reads an unsigned 32bit integer from the data.

- int **ReadSignedInteger32** ()

  Reads a signed 32bit integer from the data.

- float **ReadFloat** ()

  Reads a float from the data.

- bool **ReadBool** ()

  Reads a bool from the data.

- bool **ReadHeader** ()

  Reads the header and checks the header checksum in the length field. Returns true if the checksum is correct.

- bool **ReadLength** ()

  Reads the length and checks the header checksum in the length field. Returns true if the checksum is correct.

- bool **CheckCRC** ()

  Checks the packet checksum.

3.14.1 Detailed Description

MipBus(p. 18) Protocol Receive Message.

This class handles a received message. It contains methods to check the CRC and the header checksum, as well as methods to read the information of the message, i.e. methods that transform the received bits into integers, floats and other data types.

The documentation for this class was generated from the following file:

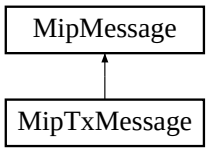
- MipRxMessage.h

3.15 MipTxMessage Class Reference

MipBus(p. 18) Protocol Transmit Message.

`#include <MipTxMessage.h>`

Inheritance diagram for MipTxMessage::



Public Member Functions

- **MipTxMessage** (int src=0, int dest=0, **eType** type=sNone)
  Constructor.
- int **GetLength** () const
  Returns the length.
- unsigned char * **GetPacket** ()
  Returns the complete packet.
- void **WriteCommand** (unsigned char value)
  Writes a command (byte) to the data.
- void **WriteByte** (unsigned char value)
  Writes a byte to the data.
- void **WriteUnsignedInteger16** (unsigned short int value)
  Writes an unsigned 16bit integer to the data.
- void **WriteSignedInteger16** (short int value)
  Writes a signed 16bit integer to the data.
- void **WriteUnsignedInteger32** (unsigned int value)
  Writes an unsigned 32bit integer to the data.
- void **WriteSignedInteger32** (int value)
  Writes a signed 32bit integer to the data.
- void **WriteFloat** (float value)
  Writes a float to the data.
- void **WriteBool** (bool value)
  Writes a bool to the data.
- void **WriteHeader** ()

Writes the message header.

- void **WriteLength** ()

  Writes the length (including the header checksum) to the message.

- void **WriteCRC** ()

  Calculates the CRC and adds it to the message.

3.15.1 Detailed Description

MipBus(p. 18) Protocol Transmit Message.

This class prepares a message for transmission to a MIP. It writes the header with the correct length and header checksum and calculates the CRC of the message. Furthermore, it contains methods to pack the arguments (integers, floats, ...) into the message.

The documentation for this class was generated from the following file:

- MipTxMessage.h

3.16 T_SysParam Struct Reference

SysParam struct.

`#include <MipDefinitions.h>`

Public Attributes

- char **AxisNumber**
  No. of the axis = MIP-Bus Address.
- char **GroupAddress**
  Group address for multicast messages.
- int **StartupMode**
  No. of active mode after startup.
- long **RS232BaudRate**
  Baudrate RS232.
- int **RS232Mode**
  0 = N81
- long **RS485BaudRate**
  Baudrate RS485.
- TSysCfg **SysConfig**
  System configuration. 16 bits: see definition below.
- TsysCfg2 **SysCongig2**
  System configuration 2. 16 bits: see definition below.
- int **SupplyVolt**
  Supply Voltage (needed for MIP-EC current regulation).
- int **NbOfChanges**
  no. of parameter changes
- int **EncoderResolution**
  encoder resolution [impulse/round]
- int **MaxVelocity**
  in [rpm]
- word **MinAccelTime**
  time [ms] to accelerate 0 to MaxVelocity
- int **PeakCurrent**
  max. peak current in [mA]

- int **ContCurrent**
  max. continuous current in [mA]
- float **GearRatio**
  total transmission ratio
- float **Inertia**
  *total inertia (rotor + load) in [g*cm2]*
- float **TorqueConst**
  in [Nm/A]
- float **SpeedConst**
  in [rpm/V] (needed for MIP-EC current regulation)
- int **CurDelta**
  third parameter for EC-current regulation
- long **PosErrorMax**
  max. position error in [qc]
- long **UpPositionLimit**
  upper position limit [qc]
- int **EnableUpLimit**
  1 = enable, 0 = disable
- long **LoPositionLimit**
  lower position limit [qc]
- int **EnableLoLimit**
  1 = enable, 0 = disable
- int **PosGainP**
  P gain for position regulation.
- int **PosGainI**
  I gain for position regulation.
- int **PosGainD**
  D gain for position regulation.
- int **CurGainP**
  P gain for current regulation.
- int **CurGainI**
  I gain for current regulation.
- int **HomeType**
  see possible types below

- float **HomeVelocity**
  velocity of home seek [qc/ms]
- int **CurrentThreshold**
  current index threshold [mA]
- long **HomeOffSet**
  relative offSet to home [qc]
- int **PLMaxVelocity**
  upper limit for pulse length velocity measurement
- int **AuxEncoderMode**
  0 = without ext. encoder, 1 = with 'a-dir' encoder, 2 = with 'a-b' encoder
- float **AuxEncoderGain**
  Auxiliary encoder gain.
- int **BrakeGain**
  low level braking gain
- int **HomePositionHi**
  position value Set after Homing (Hi-Word)
- int **HomePositionLo**
  same (Lo-Word)
- long **OneRevolution**
  one full rotor revolution [qc]
- long **OneLoadRevolution**
  one full load revolution [qc].
- word **ExtendTag**
  if == EXTEND_ TAG, following parameters exist
- long **ExtParam** [35]
  place for 35 extended parameters (43..77)
- long **SysParamCRC**
  CRC of system parameters.

3.16.1 Detailed Description

SysParam struct.

This file contains the definitions of various structs used by the MIP devices. For detailed information, please refer to the MIP manual.

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

- MipDefinitions.h

Index