Documentation

MOST High Protocol Node Layer API

Table of contents

1 Overview

This document describes the API of the CANoe nodelayer DLL “MOST_MHP.DLL”, which implements the MOST High Protocol Rev. 2.2 and 2.3. The DLL may also work with communication partners using Rev. 2.1, although it can’t be guaranteed to be fully compatible. State machine implementation is optimized for Rev 2.2 and 2.3, which implies discrepancies to the older protocol in terms of timer and retry definitions.

The DLL is compatible with CANoe.MOST 5.2 or higher.

The documentation is structured into sender and receiver functions and callbacks. Furthermore, there are callbacks which can be used in both usage contexts.

A short introduction of how to use the DLL is also provided by the “Getting started” chapter.

2 Getting started

2.1 Accessing the service

The following steps describe a quick way of how to configure a CAPL node to use the MOST High Protocol services of the DLL:

1. The nodelayer DLL should be placed in the folder of the configuration, in which the DLL will be used. Alternatively it can be placed in a folder named “Exec32” below the configuration file folder.
2. Add the DLL to the CAPL nodes that need to use MOST High Protocol transmission by selecting “Configuration…” from the context menu of the node in the setup. Click on the “Modules” tap and select the DLL by clicking on the “Add…” button.

2.2 Ports and prefixes

In its third revision, the High Protocol Nodelayer DLL is capable of managing multiple connections within a single CAPL node. This is achieved by the introduction of tx and rx ports, which enables the application to identify a connection by the unique handle of a port. So depending on how many data sources and sinks should run on a node simultaneously, the creation of a corresponding number of tx and rx ports is necessary first.

A port can only be created before measurement start by calling `MH_CreateTxPort` or `MH_CreateRxPort` within the system event handler “on prestart”. If the DLL is assigned to a CAPL test module, ports can be created anywhere within the test code, since there is no “on prestart” event handler. In this usage, the lifetime of a port is always limited to a single test module execution.

Port creation essentially assigns a user-provided prefix string to a new port handle and returns it. Depending on the port type (tx or rx), a set of callback functions can be implemented to evaluate the indications of a running connection and to manage data exchange to and from the application (for details, see section 2.3 and 2.4). The prefix string, which can also be empty in simple applications, is expected to be found at the beginning of the names of all user-implemented callback functions of a port.

Example 1 (simple usage):

```
On preStart
{
    gTxPort = MH_CreateTxPort(“”); // empty prefix
}

MH_IndTxBufferRequested(dword handle, long isPacketBegin)
{
    ...
    MH_SetTxBuffer(handle, gTxDataBuffer, bufferSize, 0);
}
...
```

Example 2 (two independent data sources):

```
On prestart
{
    gTxPort1 = MH_CreateTxPort(“Phonebook_”);   // data source port for Phonebook
    gTxPort2 = MH_CreateTxPort(“Navigation_”);  // data source port for Navigation
}
```

```
Phonebook_MH_IndTxBufferRequested(dword handle, long isPacketBegin)
{
    ...
    MH_SetTxBuffer(handle, gPhoneList, listSize, 0);
}
...
Navigation_MH_IndTxBufferRequested(dword handle, long isPacketBegin)
{
    ...
    MH_SetTxBuffer(handle, gWaypointsList, listSize, 0);
}
```

When calling `MH_CreateTxPort` or `MH_CreateRxPort`, a check for the existence of callback functions with the prefixed names is done. The measurement start may be prevented automatically in case of any missing mandatory callbacks.

In order to manage multiple connections, almost all functions have a port handle parameter to specify to which connection the function should be applied. Therefore, it is good practice to store the handles of all created ports in global variables to have them always available for calls to the DLL.

Since the handle parameter is also provided in each callback function, processing multiple connections can also be done by using only one prefix for all ports. This may reduce the number of callback functions, but also increases complexity within each callback function:

Example 3 (two data sources using a common callback infrastructure):

```
variables
{
    dword gTxPortWaypoints;
    dword gTxPortImage;
}

On prestart
{
    gTxPortWaypoints = MH_CreateTxPort("Navigation_");  // Navigation.Waypoints
    gTxPortImage     = MH_CreateTxPort("Navigation_");  // Navigation.Image
}

Navigation_MH_IndTxBufferRequested(dword handle, long isPacketBegin)
{
    if(handle == gTxPortWaypoints)
    {
        ...
        MH_SetTxBuffer(handle, gWaypointsList, listSize, 0);
    }
    else if(handle == gTxPortImage)
    {
        ...
        MH_SetTxBuffer(handle, gCurrentImageData, dataSize, 0);
    }
    ...
}
```

2.3 Data source implementation

The following section describes the basic implementation of a data source in a CAPL node. Besides the standard implementation pattern, which is relatively easy to use, but limited in its application, there is also an advanced pattern, which is more complex, but offers more flexibility in data management. Both approaches are described below.

2.3.1 Standard pattern

1. Create a tx port in the system event handler “on preStart” and store the port handle in a global variable.

```
variables
```

```
{
    dword gTxPort; // port handle is always accessible
}

on prestart
{
    gTxPort = MH_CreateTxPort("Phonebook_"); // create port and store handle
}
```

2. Optionally, transmission parameters may be adjusted before a transmission is started on the created port. This can be done by using the returned handle after creation of the port or using the special handle value of 0 before port creation. A handle value of 0 always sets the default value of a parameter in the DLL, thus all subsequently created ports apply that value automatically. The second approach is recommended, if multiple ports should use the same transmission parameters.

```
On prestart
{
    MH_SetTxTranspMode(0, 2); // set packet transport mode for all ports
    gTxPort1 = MH_CreateTxPort("Waypoints_");
    gTxPort2 = MH_CreateTxPort("Image_");

    MH_SetTxFrameSize(gTxPort1, 500);    // Set frame size of port 1
    MH_SetTxFrameSize(gTxPort2, 1000);   // Set frame size of port 2
}
```

3. Start of transmission can be triggered by calling `MH_ReqTrans` e.g. from within a keyboard event handler. Since the packet data can already be provided with that call, there's no need to implement a `MH_IndTxBufferRequested` callback.

```
On key 's'
{
    // Create a data source on port gTxPort with the provided destination addressing
    // and immediately start sending gPhoneList
    result = MH_ReqTrans(gTxPort, gDestDev, gDestFBlockID, gDestInstID, gDestFktID,
                         gDestOpTypeID, gPhoneList, listSize);
}
```

4. Optionally, implement the callback function `MH_IndTxConnectionClosed`. This is generally recommended, since it indicates the termination of a connection in a data source. The result parameter can be evaluated to find out the reason for termination (see 3.1.2.3 for details).
5. Optionally, implement the callback `MH_IndTxPktFinished`, which signals the complete transmission of a packet to the application. A call to `MH_SetTxBuffer` from within this callback starts the transmission of a new High Protocol packet over the existing connection (in contrast to a call from within `MH_IndTxBufferRequested`, which continues with the current packet).

```
Phonebook_MH_IndTxPktFinished(dword handle, long packetSize)
{
    ...
    if(IsPhoneList2Ready())
        MH_SetTxBuffer(handle, gPhoneList2, listSize, 1);
}
```

2.3.2 Advanced pattern

For simple applications that do not require dynamic transmission management, the standard pattern is sufficient. The data provided in the call to `MH_ReqTrans` is always regarded as a complete High Protocol packet.

However, if the complete data packet is not available at the beginning of the transmission, f. e. because the application generates it while transmission is in progress, packet data management can

be done dynamically by using the first signature of MH_ReqTrans in step 3, where the transmission request takes place without providing the data to be transmitted.

```
On key 's'
{
    // Create a data source on port gTxPort with the provided destination addressing
    result = MH_ReqTrans(gTxPort, gDestDev, gDestFBlockID, gDestInstID, gDestFktID,
                         gDestOpTypeID);
}
```

Consequently, the callback MH_IndTxBufferRequested, which is triggered as soon as the data source receives a positive START CONNECTION, must be implemented as the next step to provide the data. A source data buffer is provided by calling any signature of MH_SetTxBuffer. Note, that if this function is not called within the callback, the data source is automatically terminated.

Separation of the connection request from the data handling allows to control the duration of a transmission dynamically by the parameter isToBeContinued of MH_SetTxBuffer. If set to 1, after the data of the first call of MH_SetTxBuffer is transmitted, MH_IndTxBufferRequested is automatically called again to provide the next piece of data. Thus it is possible to send data in several chunks as one logical High Protocol packet. If set to 0, the data provided is regarded as the last (or single) chunk of a packet and the last block of data in that chunk will be sent as the last segment of the packet (SegID = 0x03).

```
Phonebook_MH_IndTxBufferRequested(dword handle, long isPacketBegin)
{
    ...
    // Set phone list as tx buffer; recall MH_IndTxBufferRequested after complete packet
    // in case the application has more data to send
    ...
    if (isMoreDataAvailable)
      MH_SetTxBuffer(handle, gPhoneList1, listSize, 1); // more data chunks to send
    else
      MH_SetTxBuffer(handle, gPhoneList1, listSize, 0); // current data is last chunk

}
```

The implementation of MH_IndTxConnectionClosed and MH_IndTxPktFinished follows the standard pattern as described above in steps 4 and 5.

2.4 Data sink implementation

The following section describes the basic implementation of a data sink in a CAPL node.

1. Create a rx port in the system event handler "on preStart" and store the port handle in a global variable.

```
variables
{
    dword gRxPort; // port handle is always accessible
}

on prestart
{
    gRxPort = MH_CreateRxPort("Phonebook_"); // create port and store handle
}
```

2. The capacity of a data sink port can be adjusted after creation by using the returned port handle in a call to MH_SetRxCapacity:

```
on preStart
{
    // set capacity parameters of rx port
    long framesPerBlock = 255;
    long frameSize = 256;
    long air = 1;
    long prio = 1;

    gRxPort = MH_CreateRxPort("Phonebook_"); // create port and store handle
    MH_SetRxCapacity(gRxPort, framesPerBlock, frameSize, air, prio);
}
```

3. Implement the mandatory callback function `MH_IndRequestConnection`. This callback is triggered as soon as a High Protocol transmission request is received from the node. Within the implementation, the application determines if it accepts or denies the incoming request. This may depend on the addressing and transmission parameters provided by the callback, or the current internal state of the node. Accepting an incoming request is done by calling `MH_AcceptConnection` with the handle of the rx port that will be used to host the data sink that processes the transmission. A denial of a request is done explicitly by calling `MH_DenyConnection`. If none of the both functions is called within the callback, the node will not respond at all, giving other nodes in the device the chance to process the request. Note that `MH_IndRequestConnection` is independent from any port creation, and therefore never gets a prefix in its name.

```
Variables
{
    ...
    // functional address of own DSI: FBlockID.InstID.FctID.OpType
    dword gMyFBlockID = 0x52;
    dword gMyInstID   = 0x1;
    dword gMyFktID    = 0xE01;
    dword gMyOpTypeID = 0xC;
}
...
MH_IndRequestConnection(dword dsoDevice, dword fblockID, dword instanceID, dword fktID,
dword opTypeID, dword prio, dword frameSize)
{
    if((gMyFBlockID == fblockID) &&      // check addressing ...
       (gMyInstID   == instanceID) &&
       (gMyFktID    == fktID) &&
       (gMyOpTypeID == opTypeID) &&
       (MH_GetRxPortState(gRxPortHandle) == cPortState_Inactive)) // ... and port state
    {
        MH_AcceptConnection(gRxPort);
    }
    else
    {
        MH_DenyConnection(prio + 1);
    }
}
```

4. Implement the mandatory callback function `MH_IndRxBufferRequested` for every prefix. Once a request is accepted, a data sink instance is created on the specified port. The data sink then automatically sends a START CONNECTION to the data source. `MH_IndRxBufferRequested` is triggered as soon as the data sink receives its first data frame from the source and can be used by the application to set a buffer to store the received data. It is also called during a transmission, when the provided buffer is full. Note that the only purpose of this callback is to provide a new buffer, not to read the received data into the application. This is normally done within the `MH_IndRxBufferFinished` callback that is called immediately before `MH_IndRxBufferRequested`.

```
variables
{
```

```
    ...
    const long cRxSize = 65536;
    byte gRxDataBuffer[cRxSize];
}
...
Phonebook_MH_IndRxBufferRequested(dword handle, long isPacketBegin)
{
    ...
    MH_SetRxBuffer(handle, gRxDataBuffer, cRxSize);
}
```

5. Implement the mandatory callback function `MH_IndRxBufferFinished`. This callback is triggered during a transmission, as soon as the provided buffer is full, i.e. the next block of data doesn't fit into the buffer anymore. It is also triggered, when a packet is completely received or the connection is terminated. The parameter `sizeFilled` can be used to copy the data from the buffer into the application.

```
Phonebook_MH_IndRxBufferFinished(dword handle, long sizeFilled, long reason)
{
    ...
    for(i = 0; i < sizeFilled; i++)
    {
       gPhonebook[i] = gRxDataBuffer[i];
    }
}
```

6. Implement the mandatory callback function `MH_IndRxConnectionClosed`, which is necessary for indicating the termination of a data sink. The result parameter can be evaluated to find out about the reason for termination (see 3.2.2.5 for details).

2.5 Fault injection

In addition to the normal High Protocol Services, the DLL also provides an interface for fault injection. It enables manipulation of all incoming and outgoing messages of a sender or receiver instance. The following steps describe, how to setup a custom fault injection in a CAPL node:

1. Enable fault injection for a CAPL node by calling `MH_FI_Active(1)` in the `onStart` handler.
2. Implement at least one of the callbacks `MH_FI_IndTxPreSend` and `MH_FI_IndTxPreRcv`, if the CAPL node works as a data source. Implement at least one of the callbacks `MH_FI_IndRxPreSend` and `MH_FI_IndRxPreRcv`, if the node works as a data sink. The `…PreSend` callbacks are used for manipulation of the outgoing messages, while the `…PreRcv` callbacks are used for the incoming messages.
3. Note, that all callbacks must return a value. In case of `…PreSend`, it specifies how many times the outgoing message is sent. In case of `…PreRcv`, it specifies whether the message is received (1) or not (0).
4. Within a callback, query functions such as `MH_FI_GetHighCmd`, `MH_FI_GetFrAckLow`, `MH_FI_GetFrAckHigh` and `MH_FI_GetBlockCnt` can be used to gain information about specific fields in the current message that triggered the callback. Use `MH_FI_GetLen` or `MH_FI_GetByte` to retrieve low level information.
5. Depending on the queried information, corresponding `MH_FI_Set…` functions can be used to change the data of the current message. Additionally, within the `MH_FI_IndRx…` callbacks, `MH_FI_SendFrameAck` can be used to send FRAME ACK messages, `MH_FI_SendFrameReq` can be used to send REQUEST or MULTIPLE FRAMES REQUEST messages and `MH_FI_SendAdTmRate` can be used to send ADJUST RATE messages.

3 Node Layer DLL interface

3.1 Sender

3.1.1 Functions

3.1.1.1 MH_CheckTxFile

Syntax	`long MH_CheckTxFile(char sourceFilepath[])`	
Parameter	sourceFilepath[]	Path to the file to be transmitted
Return value	Size of the file in bytes. 0, if the specified file doesn't exist.	
Purpose	Service function to determine existence and size of a file.	

3.1.1.2 MH_CreateTxPort

Syntax	`dword MH_CreateTxPort(char prefix[])`	
Parameter	Prefix	Prefix string that is expected as a prefix in the name of the callback functions used for the port to be created. Length is limited to 32 characters.
Return value	> 0: Port handle of the newly created tx port.	
	0: Port creation failed, because given prefix is too long. Measurement will not be started.	
Purpose	Creates a tx port in the node in which this function gets called. This function can only be called from within the "PreStart"-Eventhandler. The given prefix is expected to be found in front of the names of the implemented callback functions. For cases where there's no need to manage multiple connections, it is also possible to pass an empty string as prefix. The DLL looks for the implemented callback functions immediately on the call of `MH_CreateTxPort`. There are no mandatory callbacks for a tx port, but it is good practise to implement at least the following two to get some feedback about the transmission process: • *<prefix>*`MH_IndTxPktFinished(dword handle, long packetSize)` • *<prefix>*`MH_IndTxConnectionClosed(dword handle, long result)` If you want to use the `MH_ReqTrans` signature without packet data parameters to set up a data source, you must implement the following callback: • <*prefix*>MH_IndTxBufferRequested(dword handle, long isPacketBegin)	

3.1.1.3 MH_GetTxBlockAck

Syntax	`long MH_GetTxBlockAck(long handle)`	
Parameter	handle	Port handle of tx port or 0 for getting the parameter in the default parameter set of the DLL.
Return value	1: block acknowledge	
	0: frame acknowledge	
	-1: Invalid port handle	
Purpose	Returns the currently set type of acknowledge in the specified port or in the default parameter set of the DLL (handle = 0).	

3.1.1.4 MH_GetTxFrameSize

Syntax	`long MH_GetTxFrameSize(dword handle)`	
Parameter	Handle	Port handle of tx port or 0 for getting the parameter in the default parameter set of the DLL.
Return value	> 0: Frame size in bytes (max. 1006)	
	-1: Invalid port handle	
Purpose	Returns the currently set frame size of the specified tx port or of the default parameter set of the DLL (handle = 0).	

3.1.1.5 MH_GetTxPriority

Syntax	`long MH_GetTxPriority(dword handle)`	
Parameter	handle	Port handle of tx port or 0 for getting the parameter in the default parameter set of the DLL.
Return value	> 0: Transmission priority (between 1 and 127)	
	-1: Invalid port handle	
Purpose	Returns the currently set transmission priority value of the specified tx port or of the default parameter set of the DLL (handle = 0).	

3.1.1.6 MH_GetTxPortState

Syntax	`long MH_GetTxPortState(dword handle)`	
Parameter	handle	Port handle of tx port
Return value	0: Inactive (no sender instance on this node)	
	1: Data transmission	

	2: Packet data transmission finished, but END CONNECTION has not been sent yet (connection can be reused for transmission of next data packet)
	3: Sender is in the process of termination (cyclic sending of END CONNECTION)
	-1: Invalid port handle
Purpose	Returns the current state of a tx port

3.1.1.7 MH_GetTxTranspMode

Syntax	`long MH_GetTxTranspMode(dword handle)`	
Parameter	handle	Port handle of tx port or 0 for getting the parameter in the default parameter set of the DLL
Return value	1 : Control channel transmission	
	2 : Packet channel transmission	
	-1 : Invalid port handle	
Purpose	Returns the currently set transport mode of the specified tx port or of the default parameter set of the DLL (handle = 0).	

3.1.1.8 MH_ProtocolRevUsed

Syntax	`long MH_ProtocolRevUsed(dword handle)`	
Parameter	Handle	Port handle of tx port or 0 for getting the parameter in the default parameter set of the DLL
Return value	3: The nodelayer DLL uses protocol revision V. 2.3	
	2: The nodelayer DLL uses protocol revision V. 2.2	
	1: The nodelayer DLL uses protocol revision V. 2.1	
	-1: Invalid port handle	
Purpose	Returns the currently used protocol version of the specified tx port or of the default parameter set of the DLL (handle = 0).	

3.1.1.9 MH_ReqTrans

Syntax	`long MH_ReqTrans(dword handle, dword destDevice, dword fblockID, dword instID, dword fktID, dword opTypeID)`
	`long MH_ReqTrans(dword handle, dword destDevice, dword fblockID, dword instID, dword fktID, dword opTypeID, byte buffer[], long bufferSize)`

	`long MH_ReqTrans(dword handle, dword destDevice, dword fblockID, dword instID, dword fktID, dword opTypeID, char sourceFilepath[])`	
Parameter	handle	Port handle of tx port used for transmission request
	destDevice	Logical MOST device address of the destination device
	fblockID	FBlockID of High Protocol function
	instID	InstanceID of High Protocol function
	fktID	FunctionID of High Protocol function
	opTypeID	OpTypeID of High Protocol function
	buffer[]	Array containing the user data to be transmitted
	bufferSize	Length of the user data
	sourceFilepath[]	Valid file path to a file to be transmitted
Return value	0	Ok. Transmission request successfully submitted.
	-1	Invalid port handle
	-2	Port is already in use by a running connection
	-3	Signature 2 and 3: No data provided (data buffer is empty or file path is invalid)
	-4	Signature 1 of `MH_ReqTrans` requires an implementation of the callback `MH_IndTxBufferRequested` for the prefix used by the given port, but the function couldn't be found. Measurement will be stopped subsequently.
Purpose	Sends a transmission request to the specified address. Transmission parameter settings of the port specified by `handle` are applied for transport mode, priority, acknowledge type and frame size. These settings can be changed via `MH_SetTxTransportMode`, `MH_SetTxPriority`, `MH_SetTxBlockAck`, `MH_SetTxFrameSize`. The first signature separates the connection setup from the data buffer provision. Once the data sink acknowledges the transmission request by sending START CONNECTION, the callback `MH_IndTxBufferRequested` is automatically called to prompt the application to provide the first piece of data to be transmitted. This can be done by a call to `MH_SetTxBuffer`. Alternatively, the other signatures offer a quick way to provide the data to be transmitted already at the connection setup, either by passing over a data buffer (signature 2) or a path to a file. Consequently, `MH_TxBufferRequested` will not be called and transmission of data starts immediately after the connection is established.	

3.1.1.10 MH_SetTxBlockAck

Syntax	`long MH_SetBlockAck(long handle, long isBlockAck)`

Parameter	Handle	Port handle of tx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	isBlockAck	1 = block acknowledge 0 = frame acknowledge
Return value	0: Parameter successfully applied	
	-1: Invalid port handle	
Purpose	Sets the acknowledge type of the given tx port. Any connection subsequently created on that port will apply this setting.	

3.1.1.11 MH_SetTxBuffer

Syntax	`long MH_SetTxBuffer(dword handle, byte buffer[], long bufferSize, long isToBeContinued)`	
	`long MH_SetTxBuffer(dword handle, char sourceFilepath[], long isToBeContinued)`	
Parameter	handle	Port handle of tx port which hosts the connection that is requesting data to be transmitted.
	sourceFilepath[]	Path to the file to be transmitted
	isToBeContinued	Flag to indicate intention to continue transmission of further data over this connection after current data has been completely transmitted. 0: Do not call `MH_TxBufferRequested` after data has been transmitted. Provided data is transmitted as a complete High Protocol packet, which means that the last transmitted block of data will have SegID = 0x3 or 0x0, if there is only one block in the packet. 1: Call `MH_TxBufferRequested`, as soon as data has been completely transmitted. Provided data is transmitted as the beginning of a High Protocol packet, which means that the last transmitted block will have SegID = 0x1 or 0x2, if there's more than one block in the provided data.
	buffer[]	Array containing the user data to be transmitted
	bufferSize	Size of array containing the user data
Return value	>= 0: On success, signature 2 returns the size of the file in bytes, signature 1 returns 0.	
	-1: Invalid port handle	
	-2: Port currently doesn't host a running connection	
	-3: No data provided (data buffer is empty or file path is invalid)	

Purpose	Provides data to be transmitted via the data source instance on port specified by handle. This function is usually called from within the callback implementation of `MH_IndTxBufferRequested`, if the transmission of a High Protocol packet should be continued. It can also be called from outside the callback, assumed that the connection is still active, f. e. after a packet has been transmitted successfully and the application has provided more data while the connection is waiting for the expiration of $t_{DelayEnd}$ (see MOST High Protocol Specification). In this case, a call to `MH_SetTxBuffer` begins the transmission of a new packet.

3.1.1.12 MH_SetTxFrameSize

Syntax	`long MH_SetTxFrameSize(dword handle, long frameSize)`	
Parameter	handle	Port handle of tx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	frameSize	Number of user data bytes; on the control channel fixed to 11 (setting any value will be ignored); on the packet channel up to 1006 bytes
Return value	0: new frame size successfully set	
	-1: Invalid port handle	
	-3: Parameter value out of range	
Purpose	Sets the frame size of the port specified by the handle or in the default parameter set of the DLL (handle = 0). Note, that frame sizes larger than 11 bytes can only be set, if the transport mode is set to asynchronous channel beforehand by calling `MH_SetTxTranspMode`.	

3.1.1.13 MH_SetTxOnHold

Syntax	`long MH_SetTxOnHold(dword handle, long onOff, long reason)`	
Parameter	handle	Port handle of tx port
	onOff	0: Switches off cyclic sending of HOLD CONNECTION TX 1: Switches on cyclic sending of HOLD CONNECTION TX
	reason	Error code that specifies the reason for holding the connection
Return value	0: Cyclic sending successfully activated	
	-1: Invalid port handle	
	-2: Port currently doesn't host a running connection	

	-4: Not applied, since current state doesn't allow activation of cyclic sending of HOLD CONNECTION (f. e. cyclic sending is already active)
Purpose	Switches cyclic sending of HOLD CONNECTION TX on or off in the connection running on a tx port specified by handle.

3.1.1.14 MH_SetTxPriority

Syntax	`long MH_SetTxPriority(dword handle, long prio)`	
Parameter	handle	Port handle of tx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	prio	Priority of transmission (between 1 and 127)
Return value	0: New priority successfully set	
	-1: Invalid port handle	
	-3: Parameter value out of range	
Purpose	Sets the transmission priority of the port specified by the handle or in the default parameter set of the DLL (handle = 0).	

3.1.1.15 MH_SetTxTranspMode

Syntax	`long MH_SetTxTranspMode(dword handle, long transportMode)`	
Parameter	handle	Port handle of tx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	transportMode	1 : Control channel transmission 2 : Packet channel transmission
Return value	0: Transport mode successfully set	
	-1: Invalid port handle	
	-3: Parameter out of range	
Purpose	Sets the transport mode of the port specified by the handle or in the default parameter set of the DLL (handle = 0).	

3.1.1.16 MH_TxStopTrans

Syntax	`long MH_TxStopTrans(dword handle)`	
Parameter	handle	Port handle of tx port
Return value	0: Transmission successfully stopped (e.g. no connection running)	
	-1: Invalid port handle	

	-2: Port currently doesn't host a running connection
Purpose	Stops the currently running transmission on tx port specified by handle by sending END CONNECTION TX

3.1.1.17 MH_UseProtocolRev

Syntax	`long MH_UseProtocolRev(dword handle, long protocolRev)`	
Parameter	handle	Port handle of tx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	protocolRev	1: Use protocol revision V 2.1 (no revId transmitted on MOST) 2: Use protocol revision V 2.2 (revId = 1 transmitted on MOST) 3: Use protocol revision V 2.3 (revId = 2 transmitted on MOST)
Return value	0: New protocol version set	
	-1: Invalid port handle	
	-3: Parameter out of range	
Purpose	Sets the protocol version to be used in tx port specified by handle or in the default parameter set of the DLL (handle = 0). Attention: This parameter formerly was called revId. However, the coding of the protocol version was different from the revId transmitted over the bus. In order to prevent misunderstandigs this parameter was renamed to protocolRev. There is no compatbility issue, all CAPL programs still work without change. The nodelayer DLL is designed primarily with Revision V 2.2 and 2.3 in mind, so when switching to 2.1, only several aspects of the behavior are changed in order to make a sender instance compatible with a receiver device using V 2.1. Namely, the format of the REQUEST CONNECTION message is missing the RevID field and is therefore one byte shorter. Also, the SegID field is always set to 0, wherever it occurs. Note that any receiver instance always adjusts to the protocol version of the sender, through which it was invoked.	

3.1.2 Callbacks

3.1.2.1 MH_IndTxBlockFinished

Syntax	`void MH_IndTxBlockFinished(dword handle, long blockSize, long segID, long blockCnt)`	
Parameter	handle	Port handle of tx port, on which a connection has finished transmission of a block.
	blockSize	Size of transmitted block in bytes

	segID	Segment ID according to MHP Spec. Rev. 2.2 : 0x00: Single block packet transmission or older protocol version 0x01: First block of segmented transmission 0x02: Middle block of segmented transmission 0x03: Last block of segmented transmission
	blockCnt	Block sequence number
Callback type	Optional	
Purpose	Connection running on tx port referenced by handle indicates the complete transmission of a block.	

3.1.2.2 MH_IndTxBufferRequested

Syntax	`void MH_IndTxBufferRequested(dword handle, long isPacketBegin)`	
Parameter	handle	Port handle of tx port, on which a connection finished the transmission of a packet.
	isPacketBegin	0: The requested data is the continuation of a packet 1: The requested data is the beginning of a packet
Callback type	Optional (Mandatory, when using the first signature of `MH_ReqTrans`)	
Purpose	After a transmission request was performed by a call to `MH_ReqTrans` using the primary signature (lacking parameters for the data to be transmitted), the DLL asks the application to provide the data to be transmitted over the established connection by calling `MH_IndTxBufferRequested`. The application may do this by calling `MH_SetTxBuffer` in this callback. If `MH_SetTxBuffer` is called with the flag `isToBeContinued` set to 1, this callback gets triggered again after the data has been completely transmitted. This mechanism enables an application to send a High Protocol packet in several pieces. Note, that this callback is never triggered when using `MH_ReqTrans` signatures providing the data to be transmitted directly by its parameters.	

3.1.2.3 MH_IndTxConnectionClosed

Syntax	`void MH_IndTxConnectionClosed(dword handle, long result)`	
Parameter	handle	Port handle of tx port, on which a connection is closed.

	result	Error code indicating the reason for termination of transport protocol service of data source (for retry definitions see MOST High Protocol Spec. Rev 2.2): 0: Ok (Regular termination after complete transmission) 1: $r_{request}$ exceeded 2: r_{trans} exceeded 11: Connection rejected via higher priority 13: Connection terminated by receiver
Callback type	Optional	
Purpose	Indicates any kind of error occurred during establishing connection or running transmission on tx port referenced by handle. Also triggered after finished successful transmission.	

3.1.2.4 MH_IndTxPktFinished

Syntax	`void MH_IndTxPktFinished(dword handle, long packetSize)`	
Parameter	handle	Port handle of tx port, on which a connection finished the transmission of a packet
	packetSize	Size of transmitted packet in bytes
Callback type	Optional	
Purpose	Data source indicates the complete transmission of a packet. In case of a continuously streaming application, the client may call `MH_SetTxBuffer` to provide the next data packet to be transmitted within this callback implementation.	

3.1.2.5 MH_IndTxZeroFrame

Syntax	`void MH_IndTxZeroFrame(dword handle, long nOfFrames, long segID, long options, long blockCnt)`	
Parameter	handle	Port handle of tx port, on which a connection has started transmission of a block.
	nOfFrames	Number of frames to be transmitted for the current block
	segID	Segment ID according to MHP Spec. Rev. 2.2 : 0x00: Single block packet transmission or older protocol version 0x01: First block of segmented transmission 0x02: Middle block of segmented transmission 0x03: Last block of segmented transmission

	options	Options field indicating the transmission options. Bit 0-1 indicates the transport mode: 01: Block acknowledge mode 02: Frame acknowledge mode
	blockCnt	Block sequence number
Callback type	Optional	
Purpose	Connection running on tx port referenced by handle indicates the transmission of a 0-Frame	

3.2 Receiver

3.2.1 Functions

3.2.1.1 MH_AcceptConnection

Syntax	`long MH_AcceptConnection(dword handle)`	
Parameter	handle	Port handle of rx port
Return value	0: Connection request successfully assigned to rx port	
	-1: Invalid port handle	
	-2: Port is already in use by a running connection	
	-4: Execution failed, since function was called from outside the `MH_IndRequestConnection` callback.	
Purpose	Any incoming transmission request from a data source triggers the `MH_IndRequestConnection` callback. In the implementation of the callback, the application may accept the transmission by calling `MH_AcceptConnection` with the handle of the rx port, on which the new data sink should be invoked. The new data sink instance immediately sends a START CONNECTION message with the acknowledged transmission parameters back to the data source. All subsequent callbacks triggered by the transmission of the data will then use the callbacks of the assigned rx port.	

3.2.1.2 MH_CreateRxPort

Syntax	`dword MH_CreateRxPort(char prefix[])`	
Parameter	prefix	Prefix string that is expected as a prefix in the name of the callback functions used for the port to be created. Length is limited to 32 characters.
Return value	> 0: Port handle of the newly created rx port.	
	0: Port creation failed, because prefix is too long. Measurement will not be started.	

Purpose	Creates a rx port in the node in which this function gets called. This function can only be called from within the "PreStart"-Eventhandler. The given prefix is expected to be found in front of the names of the implemented callback functions. For cases where there's no need to manage multiple connections, it is also possible to pass an empty string as prefix. The DLL looks for the implemented callback functions immediately on the call of `MH_CreateRxPort`. If one of the mandatory callbacks is not implemented, the missing callback is reported in the Write-Window and measurement will not be started. Mandatory callbacks that must be implemented for a rx port: • `MH_IndRequestConnection(dword dsoDevice, dword fblockID, dword instID, dword fktID, dword opTypeID, dword prio, dword frameSize)` • `<prefix>MH_IndRxBufferRequested(dword handle, long isPacketBegin)` • `<prefix>MH_IndRxBufferFinished(dword handle, long filledSize, long reason)` • `<prefix>MH_IndRxConnectionClosed(dword handle, long result)`

3.2.1.3 MH_DenyConnection

Syntax	`long MH_DenyConnection(long priority)`	
Parameter	priority	Priority used to deny the transmission request
Return value	0: Connection request successfully denied	
	-4: Execution failed, since function was called from outside the `MH_IndRequestConnection` callback.	
Purpose	Any incoming transmission request from a data source triggers the `MH_IndRequestConnection` callback. In the implementation of the callback, the application may explicitly deny the transmission request by calling `MH_DenyConnection` with a higher priority. In case the priority value provided in the call is not higher than the one indicated by the `MH_IndRequestConnection` callback, that priority value is automatically increased by 1 and then used in the START CONNECTION response to deny connection build-up.	

3.2.1.4 MH_GetRxPriority

Syntax	`long MH_GetRxPriority(dword handle)`	
Parameter	handle	Port handle of rx port or 0 for getting the parameter in the default parameter set of the DLL
Return value	>0: Priority of transmission	
	-1: Invalid port handle	

Purpose	Returns transmission priority of rx port referenced by handle or of the default parameter set of the DLL (handle = 0).

3.2.1.5 MH_GetRxPortState

Syntax	`long MH_GetRxPortState(dword handle)`	
Parameter	handle	Port handle of rx port
Return value	0: Inactive (no sender instance on this node)	
	1: Data transmission	
	2: Data sink is in standby and ready to receive more data. This is the case immediately after the complete transmission of a packet, or when the transmission is temporarily suspended by reception of HOLD CONNECTION TX messages from the data source.	
	-1: Invalid port handle	
Purpose	Returns the current state of a rx port	

3.2.1.6 MH_MultiFrmReqUsed

Syntax	`Long MH_MultiFrmReqUsed(dword handle)`	
Parameter	handle	Port handle of rx port or 0 for getting the parameter in the default parameter set of the DLL
Return value	1: MULTIPLE FRAME REQUEST is used for requesting outstanding frames	
	0: (Single frame) REQUEST is used for requesting outstanding frames	
	-1: Invalid port handle	
Purpose	Returns the command type to be used by the receiver for requesting outstanding frames. Note that the DLL automatically requests any missing frames at the end of a block transmission, i.e. after the last frame was received.	

3.2.1.7 MH_RxStopTrans

Syntax	`long MH_RxStopTrans(dword handle)`	
Parameter	handle	Port handle of rx port
Return value	0: Transmission successfully stopped	
	-1: Invalid port handle	
	-2: Port currently doesn't host a running connection	
Purpose	Stops the currently running transmission by sending END CONNECTION RX	

3.2.1.8 MH_SetRxBuffer

Syntax	`long MH_SetRxBuffer(dword handle, BYTE buffer[], long bufferSize)`

	`long MH_SetRxBuffer(dword handle, long isToBeAppended, char destFilepath[])`	
Parameter	handle	Port handle of rx port
	buffer[]	Byte array to store received user data
	bufferSize	Size of byte array
	isToBeAppended	0: Received data is written into specified file. Any previous content will be destroyed. 1: Received data is appended at the end of the file.
	destFilepath[]	Valid file path to a file that is used to write the received user data into.
Parameter	0: New buffer successfully set	
	-1: Invalid port handle	
	-2: Port currently doesn't host a running connection	
	-3: Invalid file path or invalid value for isToBeAppended (signature 2)	
Purpose	Sets the buffer in which the data sink (on rx port specified by handle) writes the received data into. If the provided buffer is not sufficient to store at least a complete block of user data according to the acknowledged frame size and number, the connection will be rejected. Signature 2 sets the specified file as a receiver buffer. If the file doesn't exist, it is created if the folder path is valid.	

3.2.1.9 MH_SetRxCapacity

Syntax	`long MH_SetRxCapacity(dword handle, long framesPerBlock, long frameSize, long air, long priority)`	
Parameter	handle	Port handle of rx port
	framesPerBlock	Number of data frames per block
	frameSize	Size of a data frame in bytes
	air	Average interrupt rate in microseconds
	priority	Transmission priority of receiver. Allowed values range between 1 and 128.
Return value	0: New capacity successfully set	
	-1: Invalid port handle	
	-3: Single parameter out of range or combination of parameter values exceeds the maximum capacity (framesPerBlock * frameSize > 65536)	
Purpose	Sets the capacity of a rx port specified by handle.	

	By default, each rx port adjusts the capacity of a data sink created on it automatically to the requirements of the request from the data source. Calling this function explicitly sets the capacity of the data sink to given maximums. An incoming request is then acknowledged by a START CONNECTION message with scale and NDFAck values downscaled to the frames per block and frame size values given by the function call. That way it is possible to simulate devices that have limitations in their receiver capacity.

3.2.1.10 MH_SetRxOnHold

Syntax	`long MH_SetRxOnHold(dword handle, long onOff, long reason)`	
Parameter	handle	Port handle of rx port
	onOff	0: Switches off cyclic sending of HOLD CONNECTION RX != 0: Switches on cyclic sending of HOLD CONNECTION RX
	reason	Error code that specifies the reason for holding the connection
Return value	0: Cyclic sending successfully activated	
	-1: Invalid port handle	
	-2: Port currently doesn't host a running connection	
	-4: Not applied, since current state doesn't allow activation of cyclic sending of HOLD CONNECTION (f. e. cyclic sending is already active)	
Purpose	Switches cyclic sending of HOLD CONNECTION RX on or off. If MHP version 2.3 or newer is used and the function is called inside the MH_IndRxBlockFinish callback, the DLL will not send a HOLD CONNECTION message immediately but it will set the HoldFlag of the FRAME ACKNOWLEDGE message for current block to 1.	

3.2.1.11 MH_SetRxPriority

Syntax	`long MH_SetRxPriority(dword handle, long prio)`	
Parameter	handle	Port handle of rx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	prio	Priority of transmission (between 1 and 128)
Return value	0: New priority successfully set	
	-1: Invalid port handle	
	-3: Parameter out of range	
Purpose	Sets the receiver priority of rx port referenced by handle or in the default pa-	

	rameter set of the DLL (handle = 0).

3.2.1.12 MH_UseMultiFrmReq

Syntax	`long MH_UseMultiFrmReq (dword handle, long isMulti-FrameReq)`	
Parameter	handle	Port handle of rx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	isMultiFrameReq	0: Use (single frame) REQUEST for requesting outstanding frames != 0: Use MULTIPLE FRAME REQUEST for requesting outstanding frames
Return value	0: New frame request mode successfully set	
	-1: Invalid port handle	
Purpose	Sets the command type to be used by the receiver for requesting missing frames of a block. Note that the DLL automatically requests any missing frames at the end of a block transmission, i.e. after the last frame was received.	

3.2.2 Callbacks

3.2.2.1 MH_IndRequestConnection

Syntax	`void MH_IndRequestConnection(dword dsoDevice, dword fblockID, dword instID, dword fktID, dword opTypeID, dword priority, dword frameSize)`	
Parameter	dsoDevice	Address of the device in which the data source is located
	fblockID	FBlockID of the data source
	instID	InstanceID of the data source
	fktID	FunctionID of the data source
	opTypeID	OpTypeID of the data source
	priority	Priority of the transmission request
	frameSize	Suggested frame size of the data source
Callback type	Mandatory	
Purpose	This callback is triggered every time the node receives a REQUEST CONNECTION message. Addressing and parameter information is accessible via its parameter values and can be evaluated within the callback's implementation to decide, whether the current transmission request should be accepted or denied. This is done by either calling `MH_AcceptConnection` or `MH_DenyConnection`. If none of the two functions is called, the node will	

	not respond to the request.

3.2.2.2 MH_IndRxBlockFinished

Syntax	`void MH_IndRxBlockFinished(dword handle, long blockSize, long segID, long blockCnt)`	
Parameter	handle	Port handle of rx port on which a data sink finished reception of a block
	blockSize	Size of transmitted block in bytes
	segID	Segment ID according to MHP Spec. Rev. 2.2 : 0x00: Single block packet transmission or older protocol version 0x01: First block of segmented transmission 0x02: Middle block of segmented transmission 0x03: Last block of segmented transmission
	blockCnt	Block sequence number
Callback type	Optional	
Purpose	Indicates the complete reception of a block.	

3.2.2.3 MH_IndRxBufferFinished

Syntax	`void MH_IndRxBufferFinished(dword handle, long sizeFilled, long reason)`	
Parameter	handle	Port handle of rx port, on which a data sink has finished reception of a packet
	sizeFilled	Number of data bytes actually written in the buffer
	reason	0: Provided buffer is full (but more data is expected) 1: Transmission of packet is completed 2: Connection will be terminated, but there is still some received data in the buffer.
Callback type	Mandatory	
Purpose	Indicates that the provided receive buffer is filled with user data or the next transmitted block doesn't fit into the space left in the buffer. This callback is also triggered at the end of a packet or at the (forced) end of a transmission. In case the buffer was provided in form of a byte array by `MH_SetRxBuffer`, use this callback to read the received user data into the application. When a file is used as a buffer in the callback of `MH_IndRxBufferReq`, no actions need to be performed in this callback, since the file streamer cares about the data buffering automatically.	

3.2.2.4 MH_IndRxBufferRequested

Syntax	`void MH_IndRxBufferRequested(dword handle, long isPacketBegin)`	
Parameter	handle	Port handle of rx port on which a data sink needs a (new) reveiver buffer.
	isPacketBegin	0: Requested buffer is needed to continue the reception of a packet 1: Requested buffer is needed to begin the reception of a packet
Callback type	Mandatory	
Purpose	This callback is triggered as soon as the data sink sends a positive START CONNECTION message to the source or a 0-Frame is received and the data sink doesn't have enough free buffer memory available to store the next block to be transmitted. In the latter case, the callback `MH_IndRxBufferFinished` is called beforehand to notify the application that the previously received data can be read from the buffer. A new buffer or a new destination file can be set by calling `MH_SetRxBuffer`. If this function is not called within the callback implementation, the data sink automatically sends an END CONNECTION Rx to the data source.	

3.2.2.5 MH_IndRxConnectionClosed

Syntax	`void MH_IndRxConnectionClosed(dword handle, long result)`	
Parameter	handle	Port handle of rx port on which a data sink closes its connection.
	result	Result code indicating the reason for termination of transport protocol service of receiver (for retry definitions see MOST High Protocol Spec. Rev 2.2): 0: Ok (Regular termination after complete transmission) 4: r_{ready} exceeded 5: r_{negack} exceeded 10: Receiver buffer couldn't be provided 11: Connection rejected via higher priority 14: Connection terminated by sender 15: Transmission killed by sender 16: Parameter value out of range
Callback type	Mandatory	
Purpose	Indicates the termination of a connection (or the abortion of establishing a connection) on the rx port referenced by handle. The value of parameter result signals the reason for termination.	

3.2.2.6 MH_IndRxZeroFrame

Syntax	`void MH_IndRxZeroFrame(dword handle, long nOfFrames, long segID, long options, long blockCnt)`	
Parameter	handle	Port handle of rx port on which a data sink has received a 0-Frame
	nOfFrames	Number of frames in the currently transmitted block
	segID	Segment ID according to MHP Spec. Rev. 2.2 : 0x00: Single block packet transmission or older protocol version 0x01: First block of segmented transmission 0x02: Middle block of segmented transmission 0x03: Last block of segmented transmission
	options	Options field indicating the transmission options. Bit 0-1 indicates the transport mode: 01: Block acknowledge mode 02: Frame acknowledge mode
	blockCnt	Block sequence number
Callback type	Optional	
Purpose	Indicates the reception of a 0-Frame	

3.3 Common

3.3.1 Functions

3.3.1.1 MH_SetVerbose

Syntax	`void MH_SetVerbose(dword level)`	
Parameter	level	Write level output
Return value	-	
Purpose	Sets the level of verbosity for the write output of DLL. Default value is 0, which generates minimal output. If more information on events and errors occurring during operation is desired, set the level to at least 3. The DLL then generates output for all kinds of errors, which can be especially useful during development of a High Protocol application.	

3.3.2 Callbacks

3.3.2.1 MH_IndHoldRcvd

Syntax	`void MH_IndHoldRcvd(dword handle, long event)`

Parameter	handle	Port handle of tx or rx port on which a connection partner received a HOLD CONNECTION.
	event	Code specifying the reason for hold of the connection
Callback type	Optional	
Purpose	Indicates the reception of HOLD CONNECTION from the transmission partner	

3.3.2.2 MH_IndHoldSent

Syntax	`void MH_IndHoldSent(dword handle, long event)`	
Parameter	handle	Port handle of tx or rx port on which a connection partner sent a HOLD CONNECTION.
	event	Code specifying the reason for hold of the connection
Callback type	Optional	
Purpose	Indicates the transmission of a HOLD CONNECTION command	

3.3.2.3 MH_IndWrongMsg

Syntax	`void MH_IndWrongMsg(dword handle)`	
Parameter	handle	Port handle of tx or rx port on which a connection partner received a unexpected message.
Callback type	Optional	
Purpose	Indicates the reception of a message that doesn't fit to the expected communication flow. A message triggering this callback will be ignored by the state machine. In order to identify the message, the current simulation time can be retrieved from within the callback and matched with the timestamps in the trace window.	

3.4 Timeout and retry settings

3.4.1 Sender

3.4.1.1 MH_SetREnd

Syntax	`long MH_SetREnd(dword handle, long value)`	
Parameter	handle	Port handle of tx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	value	New maximum number of retries
Return value	0: value successfully set	
	-1: invalid port handle	

Purpose	Sets a new maximum number of retries for r_{End} . If value is regative, r_{End} is automatically reset to its standard value of 4.

3.4.1.2 MH_SetRRequest

Syntax	`long MH_SetRRequest(dword handle, long value)`	
Parameter	handle	Port handle of tx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	value	New maximum number of retries
Return value	0: value successfully set	
	-1: invalid port handle	
Purpose	Sets a new maximum number of retries for $r_{Request}$. If value is regative, $r_{Request}$ is automatically reset to its standard value of 4.	

3.4.1.3 MH_SetRTrans

Syntax	`long MH_SetRTrans(dword handle, long value)`	
Parameter	handle	Port handle of tx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	value	New maximum number of retries
Return value	0: value successfully set	
	-1: invalid port handle	
Purpose	Sets a new maximum number of retries for r_{Trans} . If value is regative, r_{Trans} is automatically reset to its standard value of 2.	

3.4.1.4 MH_SetTDelayEnd

Syntax	`long MH_SetTDelayEnd(dword handle, long value)`	
Parameter	handle	Port handle of tx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	value	New timeout value in milliseconds
Return value	0: value successfully set	
	-1: invalid port handle	
Purpose	Sets a new timeout value for $t_{DelayEnd}$. If value is 0, $t_{DelayEnd}$ is automatically	

	reset to its standard value of 6000 ms.

3.4.1.5 MH_SetTEnd

Syntax	`long MH_SetTEnd(dword handle, long value)`	
Parameter	handle	Port handle of tx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	value	New timeout value in milliseconds
Return value	0: value successfully set	
	-1: invalid port handle	
Purpose	Sets a new timeout value for t_{End} . If value is 0, t_{End} is automatically reset to its standard value of 100 ms.	

3.4.1.6 MH_SetTRetrans

Syntax	`long MH_SetTRetrans(dword handle, long isFrameAckMode, long value)`	
Parameter	handle	Port handle of tx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	isFrameAckMode	0: Set the new value for $t_{Retrans\_BA}$ 1: Set the new value for $t_{Retrans\_SFA}$
	value	New timeout value in milliseconds
Return value	0: value successfully set	
	-1: invalid port handle	
Purpose	Sets a new timeout value for $t_{Retrans\_BA}$ or $t_{Retrans\_SFA}$. If value is 0, $t_{Retrans\_BA}$ is automatically reset to its standard value of 200 ms; $t_{Retrans\_SFA}$ is reset to 50 ms respectively.	

3.4.1.7 MH_SetTSend

Syntax	`long MH_SetTSend(dword handle, long value)`	
Parameter	handle	Port handle of tx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	value	New timeout value in milliseconds

Return value	0: value successfully set
	-1:invalid port handle
Purpose	Sets a new timeout value for t_{Send} . If value is 0, t_{Send} is automatically reset to its standard value of 100 ms.

3.4.1.8 MH_SetTTrans

Syntax	`long MH_SetTTrans(dword handle, long value)`	
Parameter	handle	Port handle of tx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	value	New timeout value in milliseconds
Return value	0: value successfully set	
	-1: invalid port handle	
Purpose	Sets a new timeout value for t_{Trans} . If value is 0, t_{Trans} is automatically reset to its standard value of 3000 ms.	

3.4.2 Receiver

3.4.2.1 MH_SetRNegAck

Syntax	`long MH_SetRNegAck(dword handle, long value)`	
Parameter	handle	Port handle of rx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	value	New maximum number of retries
Return value	0: value successfully set	
	-1: invalid port handle	
Purpose	Sets a new maximum number of retries for r_{negack} . If value is regative, r_{negack} is automatically reset to its standard value of 8.	

3.4.2.2 MH_SetRStart

Syntax	`long MH_SetRStart(dword handle, long value)`	
Parameter	handle	Port handle of rx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.

	value	New maximum number of retries
Return value	0: value successfully set	
	-1: invalid port handle	
Purpose	Sets a new maximum number of retries for r_{Start} . If value is regative, r_{Start} is automatically reset to its standard value of 4.	

3.4.2.3 MH_SetTDwnNegAck

Syntax	`long MH_SetTDwnNegAck(dword handle, long value)`	
Parameter	handle	Port handle of rx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	value	New timeout value in milliseconds
Return value	0: value successfully set	
	-1: invalid port handle	
Purpose	Sets a new timeout value for $t_{dwnNegAck}$. If value is 0, $t_{dwnNegAck}$ is automatically reset to its standard value of 200 ms.	

3.4.2.4 MH_SetTFrame

Syntax	`long MH_SetTFrame(dword handle, long value)`	
Parameter	handle	Port handle of rx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	value	New timeout value in milliseconds
Return value	0: value successfully set	
	-1: invalid port handle	
Purpose	Sets a new timeout value for t_{Frame} . If value is 0, t_{Frame} is automatically reset to its standard value of 200 ms.	

3.4.2.5 MH_SetTReady

Syntax	`long MH_SetTReady(dword handle, long value)`	
Parameter	handle	Port handle of rx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	value	New timeout value in milliseconds

Return value	0: value successfully set
	-1: invalid port handle
Purpose	Sets a new timeout value for t_{Ready} . If value is 0, t_{Ready} is automatically reset to its standard value of 100 ms.

3.4.2.6 MH_SetTReceive

Syntax	`long MH_SetTReceive(dword handle, long value)`	
Parameter	handle	Port handle of rx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	value	New timeout value in milliseconds
Return value	0: value successfully set	
	-1: invalid port handle	
Purpose	Sets a new timeout value for $t_{Receive}$. If value is 0, $t_{Receive}$ is automatically reset to its standard value of 200 ms.	

3.4.3 Common

3.4.3.1 MH_SetTAir

Syntax	`long MH_SetTAir(dword handle, long value)`	
Parameter	handle	Port handle of tx or rx port or -1 for setting the parameter in the default rx parameter set of the DLL
	value	New timeout value in microseconds -1 for a) make the tx port use the value of AIR received by the START CONNECTION command of the receiver or b) setting the default value of the rx port.
Return value	0: value successfully set	
	-1: invalid port handle	
Purpose	Sets a new timeout value for t_{Air}. If value is -1, t_{Air} is automatically a) used as defined by the receiver or b) reset to its standard value of 5000 µs. Case a) allows overwriting the value of t_{Air} received by a sender. This applies to tx ports only, i.e. the handle must identify an existing tx port. Unlike other timer settings, this function can modify the timeout value during transmissions.	

3.4.3.2 MH_SetTHold

Syntax	`long MH_SetTHold(dword handle, long value)`

Parameter	handle	Port handle of tx or rx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	value	New timeout value in milliseconds
Return value	0: value successfully set	
	-1: invalid port handle	
Purpose	Sets a new timeout value for t_{Hold} . If value is 0, t_{Hold} is automatically reset to its standard value of 700 ms.	

3.4.3.3 MH_SetTHoldResend

Syntax	`long MH_SetTHoldResend(dword handle, long value)`	
Parameter	handle	Port handle of tx or rx port or 0 for setting the parameter in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	value	New timeout value in milliseconds
Return value	0: value successfully set	
	-1: invalid port handle	
Purpose	Sets a new timeout value for $t_{HoldResend}$. If value is 0, $t_{HoldResend}$ is automatically reset to its standard value of 500 ms.	

3.4.3.4 MH_WriteSettings

Syntax	`void MH_WriteSettings(dword handle)`	
Parameter	handle	Port handle of tx or rx port or 0 for displaying the values of the default parameter set of the DLL
Return value	-	
Purpose	Displays the current values of all adjustable timeout and retry parameters in the CANoe Write window.	

3.5 Fault Injection

3.5.1 Functions

3.5.1.1 MH_FI_Active

Syntax	`long MH_FI_Active(long active)`	
Parameter	active	0: Deactivate fault injection

		1: Activate fault injection
Return value	0: Fault injection successfully activated or deactivated	
	-4: Unable to activate fault injection, since none of the fault injection callbacks is implemented.	
Purpose	Activates or deactivates fault injection. Any of the callback functions `MH_IndTxPreSend(), MH_IndTxPreRcv(), MH_IndRxPreSend(), MH_IndRxPreRcv()` will be triggered, if it is implemented. If none of these callbacks is implemented, fault injection can't be activated.	

3.5.1.2 MH_FI_GetBlockCnt

Syntax	`long MH_FI_GetBlockCnt()`
Parameter	-
Return value	0 .. 255: Value of BlockCnt field
	-4: Not applied, since function was called from outside a fault injection callback or current message doesn't have a BlockCnt field.
Purpose	Gets the value of the BlockCnt field in a 0-frame or a frame acknowledge message.
Callbacks	All

3.5.1.3 MH_FI_GetByte

Syntax	`long MH_FI_GetByte(long position)`	
Parameter	position	Byte position between 0 and MH_FI_GetLen()-1
Return value	>= 0: Value of byte at position	
	-3: Invalid value for position parameter	
	-4: Not applied, since function was called from outside a fault injection callback.	
Purpose	Returns the value of the byte at given position in user data of message	
Callbacks	All	

3.5.1.4 MH_FI_GetFrAckHigh

Syntax	`long MH_FI_GetFrAckHigh()`
Parameter	-
Return value	0 .. 255: Value of FrAckHigh field
	-4: Not applied, since function was called from outside a fault injection callback
Purpose	Gets the value of the FrAckHigh field in a data frame or a frame acknowledge

	message.
Callbacks	All

3.5.1.5 MH_FI_GetFrAckLow

Syntax	`long MH_FI_GetFrAckLow()`
Parameter	-
Return value	0 .. 255: Value of FrAckLow field
	-4: Not applied, since function was called from outside a fault injection callback
Purpose	Gets the value of the FrAckLow field in a data frame or a frame acknowledge message.
Callbacks	All

3.5.1.6 MH_FI_GetHighCmd

Syntax	`long MH_FI_GetHighCmd()`
Parameter	-
Return value	0: The current message is a data frame
	>0: Valid high command (0xCA., 0xF2, 0xF3, 0xFC, 0xFA, 0xFB, 0xFF, 0xF0,. 0xFD, 0xF1, 0xFE)
	-4: Not applied, since function was called from outside a fault injection callback.
Purpose	Returns the value of the high command field in a control frame or 0 in case of a data frame
Callbacks	All

3.5.1.7 MH_FI_GetLen

Syntax	`long MH_FI_GetLen()`
Parameter	-
Return value	>= 0: Telegram length of current message
	-4: Not applied, since function was called from outside a fault injection callback.
Purpose	Returns the telegram length of the current message.
Callbacks	All

3.5.1.8 MH_FI_SendAdTmRate

Syntax	`long MH_FI_SendAdTmRate(long operation)`	
Parameter	operation	1: increase transmission rate

		2: decrease transmission rate
Return value	0: Message successfully sent	
	-4: Not applied, since function was called from outside the allowed fault injection callbacks.	
Purpose	Sends a ADJUST RATE message to the data source of the transmission	
Callbacks	`MH_FI_IndRxPreSend(), MH_FI_IndRxPreRcv`	

3.5.1.9 MH_FI_SendFrameAck

Syntax	`long MH_FI_SendFrameAck(long frAckHigh, long frAckLow, long blockCnt)` `long MH_FI_SendFrameAck(long frAckHigh, long frAckLow, long blockCnt, long holdFlag)`	
Parameter	frAckHigh	Value of FrAckHigh field
	frAckLow	Value of FrAckLow field
	blockCnt	Value of BlockCnt field
	holdFlag	Value of HoldFlag
Return value	0: Message successfully sent	
	-4: Not applied, since function was called from outside a fault injection callback.	
Purpose	Sends a FRAME ACK message with the specified parameters to the data source of the transmission. Set frAckLow and frAckHigh to 0 for a negative frame acknowledge. Please note that the HoldFlag field is only allowed for MHP version < 2.3 and negative frame acknowledge.	
Callbacks	`MH_FI_IndRxPreSend(), MH_FI_IndRxPreRcv`	

3.5.1.10 MH_FI_SendFrameReq

Syntax	`long MH_FI_SendFrameReq(long frAckLow)`	
	`long MH_FI_SendFrameReq(byte frameList[], long frameListLen)`	
Parameter	frAckLow	Value of FrAckLow field
	frameList	List of requested frame IDs
	frameListLen	Length of frameList
Return value	0: Message successfully sent	
	-4: Not applied, since function was called from outside the allowed fault injection callbacks.	
Purpose	1. Signature: Sends a single frame REQUEST message with the specified pa-	

	rameters to the data source of the transmission. Set frAckLow to 0 for a block request. 2. Signature: Sends a MULTIPLE FRAMES REQUEST message with the specified frame list to the data source of the transmission. Note that a receiver instance requests any outstanding data frames automatically at the end of a block transmission, i.e. after the last frame was received. So for most cases it will be sufficient to return 0 in a `MH_FI_IndRxPreRcv()` callback on reception of data frames.
Callbacks	`MH_FI_IndRxPreSend(), MH_FI_IndRxPreRcv()`

3.5.1.11 MH_FI_SetBlockCnt

Syntax	`long MH_FI_SetBlockCnt(long newValue)`	
Parameter	newValue	New value of BlockCnt field
Return value	0: New value successfully set	
	-4: Not applied, since function was called from outside a fault injection callback or current message doesn't have a FrAck field.	
Purpose	Sets the value of the BlockCnt field in a 0-frame or a frame acknowledge message.	
Callbacks	All	

3.5.1.12 MH_FI_SetByte

Syntax	`long MH_FI_SetByte(long position, long newValue)`	
Parameter	position	Byte position between 0 and MH_FI_GetLen()-1
	newValue	New byte value (0 .. 255)
Return value	0: Byte successfully set in message	
	-3: Invalid value for position parameter	
	-4: Not applied, since function was called from outside a fault injection callback	
Purpose	Sets a new byte value at given position in user data of message. Note that usage of this function may corrupt the currently received message in a way that it will not be accepted by the state machine anymore, f. e. when setting the high command byte to a unknown value. In such cases, the validation fails and the message is treated as not being received. .	
Callbacks	All	

3.5.1.13 MH_FI_SetFrAckHigh

Syntax	`long MH_FI_SetFrAckHigh(long newValue)`	
Parameter	newValue	New value of FrAckHigh field
Return value	0: New value successfully set	

	-4: Not applied, since function was called from outside a fault injection callback or current message doesn't have a FrAck field.
Purpose	Sets the value of the FrAckHigh field in a data frame or a frame acknowledge message.
Callbacks	All

3.5.1.14 MH_FI_SetFrAckLow

Syntax	`long MH_FI_SetFrAckLow(newValue)`	
Parameter	newValue	New value of FrAckLow field
Return value	0: New value successfully set	
	-4: Not applied, since function was called from outside a fault injection callback or current message doesn't have a FrAck field.	
Purpose	Sets the value of the FrAckLow field in a data frame or a frame acknowledge message.	
Callbacks	All	

3.5.1.15 MH_FI_TxBurstMode

Syntax	`long MH_FI_TxBurstMode(dword handle, long active)`	
Parameter	handle	Handle of tx port or 0 for setting burst mode in the default parameter set of the DLL. This parameter value can only be used before port creation in 'on prestart'.
	active	0: disable burst mode 1: enable burst mode
Return value	0: call unsuccessful (enable burst mode is fault injection is disabled)	
	-1: invalid port handle	
	-4: Not applied, since fault injection is disabled (call MH_FI_Active before).	
Purpose	The sender will send DATA FRAMEs as fast as possible if burst mode is activated. It won't wait for tAIR in block acknowledge mode. Once fault injection is deactivated burst mode will be deactivated automatically.	

3.5.2 Callbacks

3.5.2.1 MH_FI_IndTxPreRcv

Syntax	`long MH_FI_IndTxPreRcv(dword handle)`	
Parameter	handle	Handle of tx port, on which a data source receives a message
Return value	0: Current message is not received	

	1: Current message is received
Callback type	Optional
Purpose	Indicates the reception of a message from the data sink. Use this callback, if you want to modify these messages before they are passed on to the state machine of the data source.

3.5.2.2 MH_FI_IndTxPreSend

Syntax	`long MH_FI_IndTxPreSend(dword handle)`	
Parameter	handle	Handle of tx port, on which a data source sends a message
Return value	Specifies, how many times the current message is sent. If set to 0, the message is not sent at all.	
Callback type	Optional	
Purpose	Indicates the transmission of a message from the data source's state machine. Use this callback, if you want to modify these messages before they are put on the ring.	

3.5.2.3 MH_FI_IndRxPreRcv

Syntax	`long MH_FI_IndRxPreRcv(dword handle)`	
Parameter	handle	Handle of rx port, on which a data sink receives a message
Return value	0: Current message is not received	
	1: Current message is received	
Callback type	Optional	
Purpose	Indicates the reception of a message from the data source. Use this callback, if you want to modify these messages before they are passed on to the state machine of the data sink. Note that calling `MH_RxStopTrans` on arrival of a REQUEST CONNECTION message doesn't have any effect here, since no connection has been established at that point and therefore no data sink instance exists. See also 3.5.1.10 to force frame requests using this callback.	

3.5.2.4 MH_FI_IndRxPreSend

Syntax	`long MH_FI_IndRxPreSend(dword handle)`	
Parameter	handle	Handle of rx port, on which a data sink sends a message
Return value	Specifies, how many times the current message is sent. If set to 0, the message is not sent at all.	
Callback type	Optional	
Purpose	Indicates the transmission of a message from the data sink's state machine. Use this callback, if you want to modify these messages before they are put on the	

	ring.