DLL to interface FMod-TCP BOX

FMod-TCP BOX-DLLInterface

User Manual

Version 1.4

f i v e co
innovative engineering

Version: 1.4
Last revision: November 2nd 2011
Printed in Switzerland

Trademarks
Windows® is a registered trademark of Microsoft Corporation.
Ethernet® is a registered trademark of Xerox Corporation.
Borland® is a registered trademark of Borland Software Corporation.
Visual C++® is a registered trademark of Microsoft Corporation.
Philips® is a registered trademark of Koninklijke Philips Electronics N.V.
I2C is a registered trademark of Philips Semiconductors Corporation.

Support
e-mail: support@fiveco.ch

Table of Contents

1. Preliminary

Overview

The main goal of this Dynamic Link Library (DLL) is to help users to interface FiveCo's Ethernet products. It manages both the communication and FiveCo protocols so that you can easily read and/or write information to the next products:

- FMod-TCP BOX

This DLL is based on "C" language. It works with every compiler under "Windows 32 bits" operating system.

Files

The "FMod-TCP BOX-DLLInterface" contains 3 different Files:

FMod_TCP_BOX_DLLInterface.dll
It contains the compiled software to interface FMod-TCP BOX product.

FMod_TCP_BOX_DLLInterface.lib
This file is the library of the DLL's accessible functions. **WARNING:** this file could be specific to the compiler you use.

FMod_TCP_BOX_DLLInterface.h
This header file contains the declarations of the DLL's accessible functions.

Revision history

DLL revision	Date	Author	Note	Manual version
1.01	12.09.06	GF	- First release	1.1
1.03	27.07.07	GF	- Add scan I2C - Multiple packets - Test ping error - Synchronization + destroy threads - Mutex LogFile	1.2
1.04	13.08.07	GF	- I2C packet ID	1.3
2.00	02.11.11	GF	- FMod-TCPBOX2 compatibility - Add TIMEINSERVICE and NBPOWERUP registers - New FiveCo version management (firmware and hardware)	1.4

2. Structures of the DLL

Two structures, defined in the header file, are necessary in order to interface the DLL.

FMod_SRegisterListRead

This structure contains all the register value of the FMod-TCP BOX. It also contains Boolean value to inform the user when a register value has been read or written.
Declaration of the FMod_SRegisterListRead structure :

```
struct FMod_SRegisterListRead
{
    int  TYPE[2];         // TYPE[0] : Type / TYPE[1] : Model
    bool TYPE_Read;       // TYPE has been read

    int  VERSION_FW[2];   // Firmware version and revision
    int  VERSION_HW[2];   // Hardware version and revision
    bool VERSION_Read;    // VERSION has been read

    bool RESETCPU_Written;       // RESETCPU has been written
    bool SAVEUSERPARAMETERS_Written
    // SAVEUSERPARAMETERS has been written
    bool RESTOREUSERPARAMETERS_Written;
    // RESTOREUSERPARAMETERS has been written
    bool RESTOREFACTORYPARAMETERS_Written;
    // RESTOREFACTORYPARAMETERS has been written
    bool SAVEFACTORYPARAMETERS_Written;
    // SAVEFACTORYPARAMETERS has been written
    float     VOLTAGE;           // Input Voltage (2Bytes.2Bytes)
    bool VOLTAGE_Read;           // VOLTAGE has been read

    bool WARNINGS[32];           // Warnings (32 individual bits)
    bool WARNINGS_Read;          // WARNINGS has been read
    bool WARNINGS_Reset;         // WARNINGS has been reset

    unsigned int NBPOWERUP;      // Nb of power up
    bool NBPOWERUP_Read;         // NBPOWERUP has been read

    unsigned int TIMEINSERVICE;  // Time in service in sec
    bool TIMEINSERVICE_Read;     // TIMEINSERVICE has been read

    bool COMOPTIONS[32];         // 32 individual bits
    bool COMOPTIONS_Read;        // COMOPTIONS has been read
    bool COMOPTIONS_Written      // COMOPTIONS has been written

    int  ETHERNETMAC[6];         // Mac address (6 unsigned bytes)
    bool ETHERNETMAC_Read;       // ETHERNETMAC has been read

    int  IPADDRESS[4];           // IP Address (4 unsigned bytes)
    bool IPADDRESS_Read;         // IPADDRESS has been read
    bool IPADDRESS_Written;      // IPADDRESS has been written

    int  SUBNETMASK[4];   // Network IP subnet mask (4 unsigned bytes)
    bool SUBNETMASK_Read; // SUBNETMASK has been read
    bool SUBNETMASK_Written;     // SUBNETMASK has been written

    int  TCPTIMEOUT;             // TCP Timeout (1 unsigned byte)
    bool TCPTIMEOUT_Read;        // TCPTIMEOUT has been read
    bool TCPTIMEOUT_Written;     // TCPTIMEOUT has been written
```

```
    char MODULENAME[16];
    // Name and/or description of the module (16 unsigned bytes (char))
    bool MODULENAME_Read;         // MODULENAME has been read
    bool MODULENAME_Written;      // MODULENAME has been written

    int  RS232CONFIG;     // RS232 Baud rate (see UARTCONFIG_Enum above)
    boo  RS232CONFIG_Read;        // RS232CONFIG has been read
    bool RS232CONFIG_Written;     // RS232CONFIG has been written

    int  I2CSPD;          // I2C speed setting (see I2CSPD_Enum above)
    bool I2CSPD_Read;     // I2CSPD has been read
    bool I2CSPD_Written;  // I2CSPD has been written

    int  TCPCONNECTIONSOPENED;
    // Number ofusers connected to the card using TCP
    bool TCPCONNECTIONSOPENED_Read;
    // TCPCONNECTIONOPENED has been read

    float     ANALOGINPUTSTHRESHOLD;
    // Threshold value used by the AD converter on inputs pins
    bool ANALOGINPUTSTHRESHOLD_Read;
    // ANALOGINPUTSTHRESHOLD has been read
    bool ANALOGINPUTSTHRESHOLD_Written;
    // ANALOGINPUTSTHRESHOLD has been written

    bool OUTPUTS[16];     // Controls the state of each of the output pins.
    bool OUTPUTS_Read;    // OUTPUTS has been read
    bool OUTPUTS_Written; // OUTPUTS has been written

    bool INPUTS[16];      // Digital state of each inputs pin.
    bool INPUTS_Read;     // INPUTS has been read

    float     INPUTVOLTAGEVALUE[16];   // Voltage value of the inputs pins
    bool INPUTVOLTAGEVALUE_Read[16];
    // INPUTVOLTAGEVALUE[i] has been read

    unsigned char I2CData[I2CBUFFERSIZE]; // Array of Bytes for the I2C bus
    int  I2CData_Length;          // Length of the data in I2CData
    int  I2CData_AskID;
         // ID of the ask, specified by the user in the read function
    bool I2CData_Updated;         // I2CData has been updated

    int  NbValidI2CAdd;                // Nb of valid I2C addresses found
    unsigned char ValidI2CAdd[127];    // Array of the valid I2C addresses
    bool ValidI2CAdd_Updated;          // The scan I2C has been done

};
```

FMod_SModule

This structure contains the main information about a module. It is used to scan the network (FMod_TCP_BOX_ScanNetwork) and to receive information about all the FiveCo modules connected (not only FMod-TCP BOX) on the network.
Declaration of the FMod_SModule structure :

```
struct FMod_SModule
{
    int  TYPE[2];
    int  VERSION_FW[2];
    int  VERSION_HW[2];
    int  ETHERNETMAC[6];
    int  IPADDRESS[4];

    char MODULENAME[16];
};
```

3. Functions of the DLL

A set of functions is accessible from the FMod_TCP BOX_DLLInterface DLL to communicate with the FMod-TCP BOX product via Ethernet.

General Functions

Name of the function	Description
FMod_TCP_BOX_ScanNetwork	Retrieves a list of all FiveCo's modules connected on the network.
FMod_TCP_BOX_ChangeIPAddress	Sets the IP address and the subnet mask to the module with the specified Mac address.
FMod_TCP_BOX_VersionDLL	Gets version of the FMod-TCP BOX-DLLInterface DLL.
FMod_TCP_BOX_GetStateCommunication	Gets the actual state of the communication.
FMod_TCP_BOX_GetLastError	Gets the last error that occurred in FMod-TCP BOX-DLLInterface DLL.

MAIN Port Functions

Name of the function	Description
FMod_TCP_BOX_OpenConnection_MAINPORT	Opens the Ethernet communication on the MAIN Port.
FMod_TCP_BOX_CloseConnection	Closes the Ethernet communication.
FMod_TCP_BOX_Read_TYPE	Reads register TYPE.
FMod_TCP_BOX_Read_VERSION	Reads register VERSION.
FMod_TCP_BOX_Write_RESETCPU	Sends command RESETCPU.
FMod_TCP_BOX_Write_SAVEUSERPARAMETERS	Sends command SAVEUSERPARAMETERS.
FMod_TCP_BOX_Write_RESTOREUSERPARAMETERS	Sends command RESTOREUSERPARAMETERS.
FMod_TCP_BOX_Write_RESTOREFACTORYPARAMETERS	Sends command RESTOREFACTORYPARAMETERS.
FMod_TCP_BOX_Write_SAVEFACTORYPARAMETERS	Sends command SAVEFACTORYPARAMETERS.
FMod_TCP_BOX_Read_VOLTAGE	Reads register VOLTAGE.
FMod_TCP_BOX_Read_WARNINGS	Reads register WARNINGS.
FMod_TCP_BOX_Reset_WARNINGS	Reset register WARNINGS.

FMod_TCP_BOX_Read_NBPOWERUP	Reads register NBPOWERUP.
FMod_TCP_BOX_Read_TIMEINSERVICE	Reads register TIMEINSERVICE.
FMod_TCP_BOX_Read_COMOPTIONS	Reads register COMOPTIONS.
FMod_TCP_BOX_Write_COMOPTIONS	Writes register COMOPTIONS.
FMod_TCP_BOX_Read_ETHERNETMAC	Reads register ETHERNETMAC.
FMod_TCP_BOX_Read_IPADDRESS	Reads register IPADDRESS.
FMod_TCP_BOX_Write_IPADDRESS	Writes register IPADDRESS.
FMod_TCP_BOX_Read_SUBNETMASK	Reads register SUBNETMASK.
FMod_TCP_BOX_Write_SUBNETMASK	Writes register SUBNETMASK.
FMod_TCP_BOX_Read_TCPTIMEOUT	Reads register TCPTIMEOUT.
FMod_TCP_BOX_Write_TCPTIMEOUT	Writes register TCPTIMEOUT.
FMod_TCP_BOX_Read_MODULENAME	Reads register MODULENAME.
FMod_TCP_BOX_Write_MODULENAME	Writes register MODULENAME.
FMod_TCP_BOX_Read_RS232CONFIG	Reads register RS232CONFIG.
FMod_TCP_BOX_Write_RS232CONFIG	Writes register RS232CONFIG.
FMod_TCP_BOX_Read_I2CSPD	Reads register I2CSPD.
FMod_TCP_BOX_Write_I2CSPD	Writes register I2CSPD.
FMod_TCP_BOX_Read_TCPCONNECTIONSOPENED	Reads register TCPCONNECTIONSOPENED.
FMod_TCP_BOX_Read_ANALOGINPUTSTHRESHOLD	Reads register ANALOGINPUTSTHRESHOLD.
FMod_TCP_BOX_Write_ ANALOGINPUTSTHRESHOLD	Writes register ANALOGINPUTSTHRESHOLD.
FMod_TCP_BOX_Read_OUTPUTS	Reads register OUTPUTS.
FMod_TCP_BOX_Write_OUTPUTS	Writes register OUTPUTS.
FMod_TCP_BOX_Read_INPUTS	Reads register INPUTS.
FMod_TCP_BOX_Read_INPUTVOLTAGEVALUE	Reads one register INPUTVOLTAGEVALUE.
FMod_TCP_BOX_Read_ALL_INPUTVOLTAGEVALUE	Reads all registers INPUTVOLTAGEVALUE.
FMod_TCP_BOX_ReadWriteI2C	Reads and/or write I2C data.
FMod_TCP_BOX_ScanI2C	Scan I2C bus, looking for all connected modules.
FMod_TCP_BOX_Read_AllRegister	Reads all registers.
FMod_TCP_BOX_ResetAllRegisterRead	Erases the list of registers to be read.
FMod_TCP_BOX_SendData_MAINPORT	Prepares and sends FiveCo packets to read and/or write the specified registers

Repetitive asks Function

Name of the function	Description
FMod_TCP_BOX_RepetitiveAskStart	Starts the repetitive asks of the specified registers.
FMod_TCP_BOX_RepetitiveAskStop	Stops the repetitive asks
FMod_TCP_BOX_RepetitiveAskSetTime	Sets the time between each repetitive asks.
FMod_TCP_BOX_RepetitiveAskReset	Erases the list of registers for the repetitive asks and Stops the repetitive asks.

UART Port Functions

Name of the function	Description
FMod_TCP_BOX_OpenConnection_RS232	Opens Ethernet communication on the RS232 Port.
FMod_TCP_BOX_CloseConnection	Closes Ethernet communication.
FMod_TCP_BOX_SendData_RS232	Sends data to the connected module's RS232 port.
FMod_TCP_BOX_GetData_RS232	Receives data from the connected module's RS232 port.

4. General Functions description

FMod_TCP_BOX_ScanNetwork

```
int     FMod_TCP_BOX_ScanNetwork(FMod_SModule *ModuleArray,
            int ModuleArraySize);
```

Description

The *FMod_TCP_BOX_ScanNetwork* function retrieves the main information from all FiveCo's modules connected to the network.

Parameters

ModuleArray	[out]	Pointer on the first element of an array of FMod_SModule.
ModuleArraySize	[in]	Size of the array above.

Return Values

Number of FiveCo modules connected on the network.

Notes

You have to create an array of FMod_SModule before you call *FMod_TCP_BOX_ScanNetwork*.
If the size of the array is smaller than the number of connected modules, you will only get information on the *ModuleArraySize* firsts modules. Check that the returned value is smaller or equal to the size of the array of FMod_SModule.

FMod_TCP_BOX_ChangeIPAddress

```
bool   FMod_TCP_BOX_ChangeIPAddress(int MacAddress[6],
                int IPAddress[4], int SubnetMaskAddress[4]);
```

Description

The *FMod_TCP_BOX_ChangeIPAddress* function sets the IP address and the subnet mask of the module with the specified Mac address.

Parameters

MacAddress	[in]	Mac address of the module to change IP address.
IPAddress	[in]	IP address to be set to the module.
SubnetMaskAddress	[in]	Subnet mask to be set to the module.

Return Values

true	The function was completed successfully: both the IP address and the subnet mask are changed.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

The *FMod_TCP_BOX_ChangeIPAddress* function takes some seconds to successfully complete.

FMod_TCP_BOX_VersionDLL

```
void   FMod_TCP_BOX_VersionDLL(char* Version, int *Size);
```

Description

The *FMod_TCP_BOX_VersionDLL* function calls up the software version of the FMod_TCP_BOX_DLLInterface.dll.

Parameters

Version	[out]	Pointer on the first bytes of the buffer to receive the version.
Size	[in]	Size of the buffer *Version*.
	[out]	Nb of bytes copied in the buffer *Version*.

Return Values

No return value.

Notes

The buffer *Version* must have a minimum size of 50 bytes in order to receive the entire DLL version.

FMod_TCP_BOX_GetStateCommunication

```
int     FMod_TCP_BOX_GetStateCommunication(void *ComID);
```

Description

The *FMod_TCP_BOX_GetStateCommunication* function calls up the current state of the communication.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

A 32-bit integer corresponding to the current state of the communication specified by the *ComID* pointer:

0	The communication is closed.
1	The communication is opened.
2	An error occured in the communication.
3	The communication is opening.
4	The communication is closing.

In order to facilitate the use of *FMod_TCP_BOX_GetStateCommunication*, an enumeration of all the possible states is declared in the header file:

```
enum ComState  {  State_Closed=0, State_Opened, State_Error,
                  State_Opening, State_Closing };
```

Notes

Don't use this function to verify the open state of a connection after calling up an open connection function. The Event function Com_Event is called up for all the main communication events.

FMod_TCP_BOX_GetLastError

void FMod_TCP_BOX_GetLastError (char* Error, int *Size);

Description

The *FMod_TCP_BOX_GetLastError* function returns the last error to have occurred in the *FMod_TCP_BOX_DLLInterface.dll*.

Parameters

Error	[out]	Pointer on the first Byte of the Buffer to write the description of the last error
Size	[in]	Size of the buffer *Error*.
	[out]	Nb of bytes copied in the buffer *Error*.

Return Values

No return value.

Notes

A log file is created when an error occurs and is written to the same directory as the executable using the *FMod_TCP_BOX_DLLInterface.dll* is located.

5. MAIN Port functions descriptions

FMod_TCP_BOX_OpenConnection_MAINPORT

```
bool  FMod_TCP_BOX_OpenConnection_MAINPORT(int AddressIP[4],
            void (*Data_Received_MAINPORT)(FMod_SRegisterListRead
                  *RegList, void *ComID),
            void (*Com_Event)(int State, void *ComID),
            void **ComID);
```

Description

The *FMod_TCP_BOX_OpenConnection_MAINPORT* function opens Ethernet communication on the Main Port with a specified IP Address. Its use will return a pointer (*ComID*).

Parameters

AddressIP	[in]	IP address in an array of 32bits integer values.
Data_Received_MAINPORT	[in]	Address of the *data receive* callback function.
Com_Event	[in]	Address of the *communication event* callback function.
ComID	[out]	Pointer on the opened communication. It is the reference for this communication and is used to call up most of the functions of the *FMod_TCP_BOX_DLLInterface.dll*.

Return Values

true	The function was successfull, the communication is opening.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

The communication is really opened only when the callback function Com_Event is called with *Com_State* = 1 (*State_Opened)*.
If the communication is already opened (*ComID* not NULL), the communication will be closed and a new one created and opened. Before creating a new communication, make sure that the *ComID* pointer is NULL.

FMod_TCP_BOX_CloseConnection

bool FMod_TCP_BOX_CloseConnection (void **ComID);

Description

The *FMod_TCP_BOX_CloseConnection* function closes the Ethernet communication of the specified *ComID*.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful, the communication is closing.

Notes

The communication is closed only when the callback function Com_Event is called through *Com_State* = 0 (*State_Closed*).
If FMod_TCP_BOX_CloseConnection is called with *ComID* = NULL, it will return true because the communication is already closed. However, the function Com_Event will not be called up.

FMod_TCP_BOX_Read_TYPE

bool FMod_TCP_BOX_Read_TYPE(void *ComID);

Description

The *FMod_TCP_BOX_Read_TYPE* function prepares the DLL to read the *TYPE* (0x00) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Read_VERSION

```
bool   FMod_TCP_BOX_Read_VERSION(void *ComID);
```

Description

The *FMod_TCP_BOX_Read_VERSION* function prepares the DLL to read the *VERSION* (0x01) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Write_RESETCPU

```
bool   FMod_TCP_BOX_Write_RESETCPU(void *ComID);
```

Description

The *FMod_TCP_BOX_Write_RESETCPU* function prepares the DLL to write to the *RESETCPU* (0x02) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Write_SAVEUSERPARAMETERS

```
bool  FMod_TCP_BOX_Write_SAVEUSERPARAMETERS(void *ComID);
```

Description

The *FMod_TCP_BOX_Write_SAVEUSERPARAMETERS* function prepares the DLL to launch the *SAVEUSERPARAMETERS* (0x03) function next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Write_RESTOREUSERPARAMETERS

```
bool  FMod_TCP_BOX_Write_RESTOREUSERPARAMETERS(void *ComID);
```

Description

The *FMod_TCP_BOX_Write_RESTOREUSERPARAMETERS* function prepares the DLL to launch the *RESTOREUSERPARAMETERS* (0x04) function next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Write_RESTOREFACTORYPARAMETERS

```
bool  FMod_TCP_BOX_Write_RESTOREFACTORYPARAMETERS(
            void *ComID);
```

Description

The *FMod_TCP_BOX_Write_RESTOREFACTORYPARAMETERS* function prepares the DLL to launch the *RESTOREFACTORYPARAMETERS* (0x05) function next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Write_SAVEFACTORYPARAMETERS

```
bool   FMod_TCP_BOX_Write_SAVEFACTORYPARAMETERS(
            void *ComID);
```

Description

The *FMod_TCP_BOX_Write_SAVEFACTORYPARAMETERS* function prepares the DLL to launch the *RESTOREUSERPARAMETERS* (0x06) function next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Read_VOLTAGE

bool FMod_TCP_BOX_Read_VOLTAGE(void *ComID);

Description

The *FMod_TCP_BOX_Read_VOLTAGE* function prepares the DLL to read the *VOLTAGE* (0x07) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Read_WARNINGS

```
bool   FMod_TCP_BOX_Read_WARNINGS(void *ComID);
```

Description

The *FMod_TCP_BOX_Read_WARNINGS* function prepares the DLL to read the *WARNINGS* (0x08) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Reset_WARNINGS

```
bool  FMod_TCP_BOX_Reset_WARNINGS(void *ComID);
```

Description

The *FMod_TCP_BOX_Reset_WARNINGS* function prepares the DLL to Reset the *WARNINGS* (0x08) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

Each Bit of the WARNIGS register is set to 0.

FMod_TCP_BOX_Read_NBPOWERUP

```
bool    FMod_TCP_BOX_Read_NBPOWERUP(void *ComID);
```

Description

The *FMod_TCP_BOX_Read_NBPOWERUP* function prepares the DLL to read the NBPOWERUP (0x0B) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Read_TIMEINSERVICE

bool FMod_TCP_BOX_Read_TIMEINSERVICE(void *ComID);

Description

The *FMod_TCP_BOX_Read_TIMEINSERVICE* function prepares the DLL to read the *TIMEINSERVICE* (0x0C) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Read_COMOPTIONS

bool FMod_TCP_BOX_Read_COMOPTIONS(void *ComID);

Description

The *FMod_TCP_BOX_Read_COMOPTIONS* function prepares the DLL to read the *COMOPTIONS* (0x10) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Write_COMOPTIONS

```
bool  FMod_TCP_BOX_Write_COMOPTIONS(bool ComOpt[32],
                void *ComID);
```

Description

The *FMod_TCP_BOX_Write_COMOPTIONS* function prepares the DLL to write to the *COMOPTIONS* (0x10) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComOpt	[in]	Array of 32 Boolean value.
ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

Each Boolean value refers to one bit of the register value with the following rules:

- true → bit = 1.
- false → bit = 0.

FMod_TCP_BOX_Read_ETHERNETMAC

```
bool  FMod_TCP_BOX_Read_ETHERNETMAC(void *ComID);
```

Description

The *FMod_TCP*_BOX_*Read_ETHERNETMAC* function prepares the DLL to read the *ETHERNETMAC* (0x11) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Read_IPADDRESS

```
bool   FMod_TCP_BOX_Read_IPADDRESS(void *ComID);
```

Description

The *FMod_TCP_BOX_Read_IPADDRESS* function prepares the DLL to read the *IPADDRESS* (0x12) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Write_IPADDRESS

```
bool   FMod_TCP_BOX_Write_IPADDRESS(int IPAdd[4], void *ComID);
```

Description

The *FMod_TCP_BOX_Write_IPADDRESS* function prepares the DLL to write to the *IPADDRESS* (0x12) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

IPAdd	[in]	Array of 4 "32 bits Integer value".
ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

The IPAdd array must contain valid value for an IP address.
$0 \leq$ value ≤ 255.

FMod_TCP_BOX_Read_SUBNETMASK

```
bool   FMod_TCP_BOX_Read_SUBNETMASK(void *ComID);
```

Description

The *FMod_TCP_BOX_Read_SUBNETMASK* function prepares the DLL to read the *SUBNETMASK* (0x13) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Write_SUBNETMASK

```
bool   FMod_TCP_BOX_Write_SUBNETMASK(int SubnetMask[4],
                void *ComID);
```

Description

The *FMod_TCP_BOX_Write_SUBNETMASK* function prepares the DLL to write to the *SUBNETMASK* (0x13) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

SubnetMask	[in]	Array of 4 "32 bits Integer value".
ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

The SubnetMask array must contain valid value for an IP address.
$0 \leq$ value ≤ 255.

FMod_TCP_BOX_Read_TCPTIMEOUT

```
bool   FMod_TCP_BOX_Read_TCPTIMEOUT(void *ComID);
```

Description

The *FMod_TCP_BOX_Read_TCPTIMEOUT* function prepares the DLL to read the *TCPTIMEOUT* (0x14) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Write_TCPTIMEOUT

```
bool   FMod_TCP_BOX_Write_TCPTIMEOUT(int TCPTimeOut,
                void *ComID);
```

Description

The *FMod_TCP_BOX_Write_TCPTIMEOUT* function prepares the DLL to write to the *TCPTIMEOUT* (0x14) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

TCPTimeOut	[in]	32 bits Integer value.
ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

The TCPTimeOut value must contain a valid value between 0 and 255.

FMod_TCP_BOX_Read_MODULENAME

```
bool  FMod_TCP_BOX_Read_MODULENAME(void *ComID);
```

Description

The *FMod_TCP_BOX_Read_MODULENAME* function prepares the DLL to read the *MODULENAME* (0x15) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Write_MODULENAME

```
bool  FMod_TCP_BOX_Write_MODULENAME(char Name[16],
                void *ComID);
```

Description

The *FMod_TCP_BOX_Write_MODULENAME* function prepares the DLL to write to the *MODULENAME* (0x15) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

Name	[in]	Array of 16 char (1 Byte) value.
ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Read_RS232CONFIG

bool FMod_TCP_BOX_Read_RS232CONFIG(void *ComID);

Description

The *FMod_TCP_BOX_Read_RS232CONFIG* function prepares the DLL to read the *RS232CONFIG* (0x16) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details..

FMod_TCP_BOX_Write_RS232CONFIG

bool FMod_TCP_Write_RS232CONFIG(int Config, void *ComID);

Description

The *FMod_TCP_BOX_Write_RS232CONFIG* function prepares the DLL to write to the *RS232CONFIG* (0x16) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

Config	[in]	32 bits Integer value.
ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

An enumerator in the header file *FMod_TCP_BOX_DLLInterface.h* lists all the possible values of Config :

```
enum RS232CONFIG_Enum
{
    Spd_4800Bps=0,
    Spd_9600Bps,
    Spd_19200Bps,
    Spd_38400Bps,
    Spd_57600Bps,
    Spd_115200Bps,
    Spd_4800Bps_FC=128,
    Spd_9600Bps_FC,
    Spd_19200Bps_FC,
    Spd_38400Bps_FC,
    Spd_57600Bps_FC,
    Spd_115200Bps_FC
};
```

FMod_TCP_BOX_Read_I2CSPD

```
bool   FMod_TCP_BOX_Read_I2CSPD(void *ComID);
```

Description

The *FMod_TCP_BOX_Read_I2CSPD* function prepares the DLL to read the *I2CSPD* (0x18) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Write_I2CSPD

bool FMod_TCP_BOX_Write_I2CSPD(int Spd, void *ComID);

Description

The *FMod_TCP_BOX_Write_I2CSPD* function prepares the DLL to write to the *I2CSPD* (0x18) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

Spd	[in]	32 bits Integer value.
ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

An enumerator in the header file *FMod_TCP_BOX_DLLInterface.h* lists all the possible values of Spd:

```
enum    I2CSPD_Enum
{
    Spd_100KHz=99,
    Spd_400KHz=24,
};
```

FMod_TCP_BOX_Read_TCPCONNECTIONSOPENED

bool FMod_TCP_BOX_Read_TCPCONNECTIONSOPENED(void *ComID);

Description

The *FMod_TCP_BOX_Read_TCPCONNECTIONSOPENED* function prepares the DLL to read the *TCPCONNECTIONSOPENED* (0x1A) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Read_ANALOGINPUTSTHRESHOLD

```
bool  FMod_TCP_BOX_Read_ANALOGINPUTSTHRESHOLD (
            void *ComID);
```

Description

The *FMod_TCP_BOX_Read_ANALOGINPUTSTHRESHOLD* function prepares the DLL to read the *ANALOGINPUTSTHRESHOLD* (0x20) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Write_ ANALOGINPUTSTHRESHOLD

```
bool  FMod_TCP_BOX_Write_ANALOGINPUTSTHRESHOLD(
            float AnalogValue,  void *ComID);
```

Description

The *FMod_TCP_BOX_Write_ANALOGINPUTSTHRESHOLD* function prepares the DLL to write to the *ANALOGINPUTSTHRESHOLD* (0x20) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

AnalogValue	[in]	32 bits floating-point data value.
ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

The *AnalogValue* must be a floating point value between -12 and 12.

FMod_TCP_BOX_Read_OUTPUTS

```
bool  FMod_TCP_BOX_Read_OUTPUTS(void *ComID);
```

Description

The FMod_TCP_BOX_Read_OUTPUTS function prepares the DLL to read the *OUTPUTS* (0x21) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_Write_OUTPUTS

```
bool   FMod_TCP_BOX_Write_OUTPUTS(bool Outputs[16],
                  void *ComID);
```

Description

The *FMod_TCP_BOX_Write_OUTPUTS* function prepares the DLL to write to the *OUTPUTS* (0x21) register next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

Outputs	[in]	Array of 16 Boolean value.
ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

Each Boolean value refers to an I/O direction:

- true → bit = 1 : Input.
- false → bit = 0 : Output.

FMod_TCP_BOX_Read_INPUTS

```
bool   FMod_TCP_BOX_Read_INPUTS(bool Continuous, void *ComID);
```

Description

The *FMod_TCP_BOX_Read_INPUTS* function prepares the DLL to read the *INPUTS* (0x23) register next time FMod_TCP_BOX_SendData_MAINPORT is called or to add it in the list of registers used for repetitive ask.

Parameters

Continuous	[in]	Boolean for repetitive ask of this register.
ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

Continuous = false	→	The register will be read next time FMod_TCP_BOX_SendData_MAINPORT is called.
Continuous = true	→	The register will be read repetitively as soon as the FMod_TCP_BOX_RepetitiveAskStart is called.

FMod_TCP_BOX_Read_INPUTVOLTAGEVALUE

```
bool   FMod_TCP_BOX_Read_INPUTVOLTAGEVALUE (int InputNb,
                bool Continuous, void *ComID);
```

Description

The *FMod_TCP_BOX_Read_INPUTVOLTAGEVALUE* function prepares the DLL to read one of the *INPUTVOLTAGE**X**VALUE* (0x30 to 0x3F) register next time FMod_TCP_BOX_SendData_MAINPORT is called or to add it in the list of registers used for repetitive ask.

Parameters

InputNb	[in]	32 bits Integer value, input pin number
Continuous	[in]	Boolean for repetitive ask of this register.
ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

Continuous = false	→	The register will be read next time FMod_TCP_BOX_SendData_MAINPORT is called.
Continuous = true	→	The register will be read repetitively as soon as the FMod_TCP_BOX_RepetitiveAskStart is called.

InputNb specify the number of the input to read. It must be an integer value between 0 and 15.

FMod_TCP_BOX_Read_ALL_INPUTVOLTAGEVALUE

```
bool   FMod_TCP_BOX_Read_ALL_INPUTVOLTAGEVALUE (
            bool Continuous, void *ComID);
```

Description

The *FMod_TCP_BOX_Read_ALL_INPUTVOLTAGEVALUE* function prepares the DLL to read all the *INPUTVOLTAGE**X**VALUE* (0x30 to 0x3F) registers next time FMod_TCP_BOX_SendData_MAINPORT is called or to add them in the list of registers used for repetitive ask.

Parameters

Continuous	[in]	Boolean for repetitive ask of this register.
ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

Continuous = false	→	The registers will be read next time FMod_TCP_BOX_SendData_MAINPORT is called.
Continuous = true	→	The registers will be read repetitively as soon as the FMod_TCP_BOX_RepetitiveAskStart is called.

FMod_TCP_BOX_ReadWriteI2C

```
bool   FMod_TCP_BOX_ReadWriteI2C(unsigned char *DataBuf,
                int DataLength, bool Continuous , int AskID, void *ComID);
```

Description

The *FMod_TCP_BOX_ReadWriteI2C* function prepares the DLL to read or/and write data on the I2C Bus next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

DataBuf	[in]	Pointer on the I2C data.
DataLength	[in]	Length of the I2C data.
Continuous	[in]	Boolean for repetitive asks of this I2C data.
AskID	[in]	ID for this I2C ask.
ComID	[in]	Pointer on the communication.

Return Values

true	The function successes.
false	The function fails. See log file or call FMod_TCP_BOX_GetLastError to have error details.

Notes

The I2C data must follow the next sequence (see FMod-TCP BOX user's manual for more details):

Byte#		Number of bits	Example
0x00	7 bits Address (bit 7 = 0)	8 bits	0x28
0x01	X (number of bytes to write)	8 bits	0x02
0x02	xBytes	X bytes	0xAF1D
....	Y (number of bytes to read)	8 bits	0x05
The four previous entries can be replicated to access the same or other I2C slaves within this command sequence.			

The AskID variable allows the user to identify the answer from the module. When an I2C repetitive ask is running and a normal ask is made, the answer from the module is identified through this ID. For repetitive ask, the answer ID is always the same. A classic way to use this variable is to define one value for the repetitive ask and another for the other asks.

FMod_TCP_BOX_ScanI2C

```
bool   FMod_TCP_BOX_ScanI2C(void *ComID);
```

Description

The *FMod_TCP_BOX_ScanI2C* function prepares the DLL for a test of all available addresses to check I2C modules availability next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function successes.
false	The function fails. See log file or call FMod_TCP_BOX_GetLastError to have error details.

Notes

The available modules addresses will be in the ValidI2CAdd[127] array. The NbValidI2CAdd value is the number of I2C modules found.

FMod_TCP_BOX_Read_AllRegister

```
bool   FMod_TCP_BOX_Read_AllRegister(void *ComID);
```

Description

The *FMod_TCP_BOX_Read_AllRegister* function prepares the DLL to read all the registers of the module next time FMod_TCP_BOX_SendData_MAINPORT is called.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_ResetAllRegisterRead

```
bool  FMod_TCP_BOX_ResetAllRegisterRead(void *ComID);
```

Description

The *FMod_TCP_BOX_ResetAllRegisterRead* function resets the DLL of all registers set to be read.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_SendData_MAINPORT

```
bool   FMod_TCP_BOX_SendData_MAINPORT(void *ComID);
```

Description

The *FMod_TCP_BOX_SendData_MAINPORT* function sends a FiveCo packet to the device with all of the user's requests.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful, data is sent.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

Sent data (read values and/or write acknowledge) results in a calling up of the callback function Data_Received_MAINPORT.

6. Repetitive ask Function

This set of function allows you to read some registers repetitively within a specified delay.
For example, if you want to read the register INPUTS every 10[ms], you only have to follow the next steps:

1. Set the register INPUTS in continuous read mode:
   Call FMod_TCP_BOX_Read_INPUTS with continuous variable = true,
2. Set the repetitive time to 10[ms]
   Call FMod_TCP_BOX_RepetitiveAskSetTime with TimeMiliSec = 10.
3. Start the repetitive ask
   Call FMod_TCP_BOX_RepetitiveAskStart

Each 10[ms], the callback function Data_Received_MAINPORT will then be called up with the updated value for the register INPUTS.
To stop the repetitive ask, just call FMod_TCP_BOX_RepetitiveAskStop.
To add another register to the list for repetitive asks, you have to call the FMod_TCP_BOX_Read_XXX function with continuous = true. You can perform this task even if the repetitive ask is already running.

FMod_TCP_BOX_RepetitiveAskStart

```
bool  FMod_TCP_BOX_RepetitiveAskStart(void *ComID);
```

Description

The *FMod_TCP_BOX_RepetitiveAskStart* start the automatic repetitive ask of specified registers.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

To specify which register (XXX) will be read, use FMod_TCP_BOX_Read_XXX function with continuous = true.
You can also erase the list of registers for repetitive asks with the function FMod_TCP_BOX_RepetitiveAskReset.
To set the time of the repetition, call up FMod_TCP_BOX_RepetitiveAskSetTime.
The default time value between two automatic repetitive asks is 100[ms].

FMod_TCP_BOX_RepetitiveAskStop

```
bool   FMod_TCP_BOX_RepetitiveAskStop(void *ComID);
```

Description

The *FMod_TCP_BOX_RepetitiveAskStop* stops the automatic repetitive asks of specified registers.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes (note?)

These function only stops the loop of repetitive asks. The list of registers to read remains unchanged.

FMod_TCP_BOX_RepetitiveAskSetTime

```
bool  FMod_TCP_BOX_RepetitiveAskSetTime(int TimeMiliSec,
                void *ComID);
```

Description

The *FMod_TCP_BOX_RepetitiveAskSetTime* sets the delay between two automatic repetitive asks.

Parameters

TimeMiliSec	[in]	Delay in milliseconds.
ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

The TimeMiliSec must be positive and greater than zero.
The smallest value for this time is 1[ms]. The DLL can follow this repetitive time value if the CPU is powerful enough and if the network to the connected module is fast enough.

FMod_TCP_BOX_RepetitiveAskReset

```
bool   FMod_TCP_BOX_RepetitiveAskReset(void *ComID);
```

Description

The *FMod_TCP_BOX_RepetitiveAskReset* erases the list of registers for the automatic repetitive ask and stops the repetitive ask.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

7. UART Port Functions

FMod_TCP_BOX_OpenConnection_RS232

```
bool   FMod_TCP_OpenConnection_RS232(int AddressIP[4],
                    void (*Data_Received_RS232)(int NbByte, void *ComID),
                    void (*Com_Event)(int State, void *ComID),
                    void **ComID);
```

Description

The *FMod_TCP_BOX_OpenConnection_RS232* function opens the Ethernet communication with the specified IP Address on the RS232 Port. It returns a pointer (*ComID*) to be used for this communication.

Parameters

AddressIP	[in]	IP address in an array of 32bits integer values.
Data_Received_RS232	[in]	Address of the data received callback function.
Com_Event	[in]	Address of the communication event callback function.
ComID	[out]	Pointer on the opened communication. It's the reference of this communication and is used to call most of the *FMod_TCP BOX DLLInterface* functions.

Return Values

true	The function is successful, communication is opening
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

The communication is really opened only when the callback function Com_Event is called with *Com_State* = 1 (*State_Opened)*.
If the communication is already opened (*ComID* not NULL), the communication will be closed and a new one created and opened. Make sure before opening a new communication that the *ComID* pointer is NULL.

FMod_TCP_BOX_CloseConnection

bool FMod_TCP_BOX_CloseConnection (void **ComID);

Description

The *FMod_TCP_BOX_CloseConnection* function closes the Ethernet communication of the specified *ComID*.

Parameters

ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful, the communication is closing.

Notes

The communication is really closed only when the callback function Com_Event is called with *Com_State* = 0 (*State_Closed*).
If FMod_TCP_BOX_CloseConnection is called with *ComID* = NULL, it will return true because the communication is already closed. However, the function Com_Event will not be called.

FMod_TCP_BOX_SendData_RS232

```
bool   FMod_TCP_BOX_SendData_RS232(unsigned char *DataBuf,
                int DataLength, void *ComID);
```

Description

The *FMod_TCP_BOX_SendData_RS232* function sends data to the RS232 bus.

Parameters

DataBuf	[in]	Pointer on the RS232 data.
DataLength	[in]	Length of the RS232 data.
ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

FMod_TCP_BOX_GetData_RS232

```
bool   FMod_TCP_BOX_GetData_RS232(unsigned char *DataBuf,
                int *DataLength, void *ComID);
```

Description

The *FMod_TCP_BOX_GetData_RS232* reads the received data from the RS232 bus.

Parameters

DataBuf	[in]	Pointer on the RS232 buffer.
DataLength	[in]	Size of the RS232 buffer.
	[out]	Length of received RS232 data.
ComID	[in]	Pointer on the communication.

Return Values

true	The function is successful.
false	The function failed. See log file or call FMod_TCP_BOX_GetLastError to get error details.

Notes

This function should only be used when the callback function Data_Received_RS232 is called up by the DLL.

8. Communication events functions

The *FMod_TCP_BOX_DLLInterface.dll* is based on event interaction. This way, one doesn't have to call a function to know if data has been received. You only have to wait for the call of one of the next callback functions to know about important events in the communication process, such as data being received or the main states of the communication (ie: closed, opened or error).
The following functions are given in parameters of the next functions:

- FMod_TCP_BOX_OpenConnection_MAINPORT
- FMod_TCP_BOX_OpenConnection_RS232

Name of the function	Description
Data_Received_MAINPORT	This function is called when data is received from the connected module on the Main Port.
Data_Received_RS232	This function is called when data is received from the connected module on the RS232 Port.
Com_Event	This function is called to inform user of a major communication event.

It is recommended to use a different Data_Received_MAINPORT, Data_Received_RS232 and Com_Event function for each opened communication in order to facilitate the reception of information from the DLL.

Data_Received_MAINPORT

```
void   Data_Received_MAINPORT(FMod_SRegisterListRead*RegList,
                              void *ComID)
```

Description

The *Data_Received_MAINPORT* function is called by the DLL when data received from the connected module on the Main Port is ready to be read.

Parameters

RegList	[in]	Pointer on the structure containing the module's information.
ComID	[in]	Pointer on the communication.

Return Values

No return value.

Notes

The Pointer on the structure is at your disposal during this event. When this function returns, the DLL could change the data of this structure when new data is incoming. Therefore, you have to read and/or store data before returning this event function.
Until the user returns this function, the communication will be blocked even if a repetitive ask has been activated.

Data_Received_RS232

void Data_Received_RS232(int NbByte, void *ComID)

Description

The *Data_Received_RS232* function is called by the DLL when data received from the connected module on the RS232 Port is ready to be read with FMod_TCP_BOX_GetData_RS232.

Parameters

NbByte	[in]	Number of bytes to read.
ComID	[in]	Pointer on the communication.

Return Values

No return value.

Notes

The buffer of received data is at your disposal during this event. When this function returns, the DLL could change or erase the buffer if new data are incoming. Therefore, you have to read data before returning this event function with FMod_TCP_BOX_GetData_RS232.
Until the user returns this function, the communication will be blocked even if a repetitive ask has been activated.

Com_Event

void ComEvent(int State, void *ComID);

Description

The *ComEvent* function is called by the DLL when a main event occurs during the communication process.

Parameters

State	[in]	Actual state of the communication.
ComID	[in]	Pointer on the communication.

Return Values

No return value.

Notes

This event informs the user of the TCP communication's next events:

- The communication is opened *State = 1 (State_Opened)*
- The communication is closed *State = 0 (State_Closed)*
- An error occurred during communication *State = 2 (State_Error)*

The pointer *ComID* on the communication received by this event function is a copy of the one received *by the functions:*

- FMod_TCP_BOX_OpenConnection_MAINPORT
- FMod_TCP_BOX_OpenConnection_RS232

You can use this pointer to call up any FMod_TCP_BOX_DLLInterface DLL function. However, if the function modifies the pointer *ComID* (like FMod_TCP_BOX_CloseConnection), you will have to copy the result value in your own *ComID* pointer for this communication.

9. Applications example

MAIN Port : Open and close communication

Header File

```
#include " FMod_TCP_BOX_DLLInterface.h"

//---------------------------------------------------------------------------
//  MAIN port communication parameters
//---------------------------------------------------------------------------

void *ComID_MAINPORT;

// Callback functions
void Data_Received_MAINPORT(FMod_SRegisterListRead *RegList, void *ComID);
void ComEvent_MAINPORT(int State, void *ComID);
```

Source File

```
//---------------------------------------------------------------------------
//  MAIN port communication callback : Data Receive
//---------------------------------------------------------------------------
void Data_Received_MAINPORT (FMod_SRegisterListRead *RegList, void *ComID);
{

}
//---------------------------------------------------------------------------
//  MAIN port communication callback : Communication Event
//---------------------------------------------------------------------------
void ComEvent_MAINPORT (int State, void *ComID)
{

}
//---------------------------------------------------------------------------
//  MAIN port open communication
//---------------------------------------------------------------------------
void OpenConnection_MainPORT( )
{
    int add[4] = {169, 254, 5, 5};
    ComID_MAINPORT = NULL;

    FMod_TCP_BOX_OpenConnection_MAINPORT (add, Data_Received_MAINPORT,
                                                ComEvent_MAINPORT,
                                                &ComID_MAINPORT);
}
//---------------------------------------------------------------------------
//  MAIN port close communication
//---------------------------------------------------------------------------
void CloseConnection_MainPORT ( )
{
    FMod_TCP_BOX_CloseConnection(&ComID_MAINPORT);
}
//---------------------------------------------------------------------------
```

MAIN Port communication: Read and Write register

```
//-----------------------------------------------------------------------------
//  MAIN port communication callback : Data Receive
//-----------------------------------------------------------------------------
void Data_Received_MAINPORT (FMod_SRegisterListRead *RegList, void *ComID);
{
    int tcpTime;

    if(RegList->TCPTIMEOUT_Read)
    {
         tcpTime = RegList->TCPTIMEOUT;
    }

    if(RegList->TCPTIMEOUT_Written)
    {
         // acknowledge of TCPTIMEOUT register write
    }
}
//-----------------------------------------------------------------------------
//  MAIN port communication : Read register TCPTIMEOUT
//-----------------------------------------------------------------------------
void Read_TCPTIMEOUT ( );
{
    // Prepare to read Register TCPTIMEOUT
    FMod_TCP_BOX_Read_TCPTIMEOUT(ComID_MAINPORT);

    // Makes and send the packet to the module
    FMod_TCP_BOX_SendData_MAINPORT(ComID_MAINPORT);
}
//-----------------------------------------------------------------------------
//  MAIN port communication : Write register TCPTIMEOUT
//-----------------------------------------------------------------------------
void Write_TCPTIMEOUT( );
{
    // Prepare to write Register TCPTIMEOUT (value = 25sec)
    FMod_TCP_BOX_Write_TCPTIMEOUT(25, ComID_MAINPORT);

    // Makes and send the packet to the module
    FMod_TCP_BOX_SendData_MAINPORT(ComID_MAINPORT);
}
//-----------------------------------------------------------------------------
//  MAIN port communication : Read all registers
//-----------------------------------------------------------------------------
void Read_ALLREGISTER( );
{
    // Prepare to read all Registers
    FMod_TCP_BOX_Read_AllRegister(ComID_MAINPORT);

    // Makes and send the packet to the module
    FMod_TCP_BOX_SendData_MAINPORT(ComID_MAINPORT);
}
```

MAIN Port communication: Repetitive register read

```
//-----------------------------------------------------------------------------
//  MAIN port communication callback : Data Receive
//-----------------------------------------------------------------------------
void Data_Received_MAINPORT (FMod_SRegisterListRead *RegList, void *ComID);
{
    bool INPUTS[16];

    if(RegList->INPUTS_Read)
    {
         for(int i=0; i<16; i++)
              INPUTS = RegList-> INPUTS[i];
    }
}
//-----------------------------------------------------------------------------
//  MAIN port communication : Read register INPUTS in continuous
//-----------------------------------------------------------------------------
void Start_Read_INPUTS_Rep( );
{
    // Prepare to read Register INPUTS in continuous
    FMod_TCP_BOX_Read_INPUTS(true, ComID_MAINPORT);

    // Set repetitive time to 10[ms]
    FMod_TCP_BOX_RepetitiveAskSetTime(10, ComID_MAINPORT);

    // Start the repetitive ask
    FMod_TCP_BOX_RepetitiveAskStart(ComID_MAINPORT);
}
//-----------------------------------------------------------------------------
//  MAIN port communication : Stop repetitive ask
//-----------------------------------------------------------------------------
void Stop_Read_INPUTS_Rep( );
{
    // Start the repetitive ask
    FMod_TCP_BOX_RepetitiveAskStop(ComID_MAINPORT);
}
```

MAIN Port communication: Send I2C data

```
//-----------------------------------------------------------------------------
//  MAIN port communication callback : Data Receive
//-----------------------------------------------------------------------------
void Data_Received_MAINPORT (FMod_SRegisterListRead *RegList, void *ComID);
{
    if(RegList->I2CData_Updated)
    {
         // Read I2CData buffer
    }
}
//-----------------------------------------------------------------------------
//  MAIN port communication : Send I2C data
//-----------------------------------------------------------------------------
void Send_I2CData( );
{
    unsigned char I2CBuffer[10];
    // … Fill the buffer with I2C data

    // Prepare to send I2C data
    FMod_TCP_BOX_ReadWriteI2C (I2CBuffer , 10, true, ComID_MAINPORT);

    // Makes and send the packet to the module
    FMod_TCP_BOX_SendData_MAINPORT(ComID_MAINPORT);
}
```

RS232 Port communication

Header File

```
#include "FMod_TCP_BOX_DLLInterface.h"

//---------------------------------------------------------------------------
//  RS232 port communication parameters
//---------------------------------------------------------------------------
void *ComID_RS232;

unsigned char ReadBuffer[256];

// Callback functions
void DataReceived_RS232(int NbByte, void *ComID);
void ComEvent_RS232(int State, void *ComID);
```

Source File

```
//---------------------------------------------------------------------------
//  RS232 port communication callback : Data Receive
//---------------------------------------------------------------------------
void DataReceived_RS232 ((int NbByte, void *ComID);
{
    // Check if ReadBuffer is big enough to receive NbByte Bytes
    int Length = 256;
    FMod_TCP_BOX_GetData_RS232(ReadBufferRS232, &Length, ComID_RS232);

    // Length contain now the real number of Bytes copied in ReadBuffer
}
//---------------------------------------------------------------------------
//  RS232 port communication callback : Communication Event
//---------------------------------------------------------------------------
void ComEvent_RS232 (int State, void *ComID)
{
}
//---------------------------------------------------------------------------
//  RS232 port open communication
//---------------------------------------------------------------------------
void OpenConnection_RS232PORT( )
{
    int add[4] = {169, 254, 5, 5};
    ComID_RS232 = NULL;

    FMod_TCP_BOX_OpenConnection_RS232(add, DataReceived_RS232,
                                      ComEvent_RS232, &ComID_RS232);
}
//---------------------------------------------------------------------------
//  UART port close communication
//---------------------------------------------------------------------------
void CloseConnection_RS232PORT ( )
{
    FMod_TCP_BOX_CloseConnection(&ComID_RS232);
}
//---------------------------------------------------------------------------
//  UART port Send data
//---------------------------------------------------------------------------
void CloseConnection_RS232PORT ( )
{
    unsigned char DataBuf[10] = "Test UART";
    FMod_TCP_BOX_SendData_RS232(DataBuf, 9, ComID_RS232);
}
//---------------------------------------------------------------------------
```

Contact address :

FiveCo - Innovative Engineering
PSE-C
CH-1015 Lausanne
Switzerland
Tel: +41 21 693 86 71
Fax: +41 21 693 86 70

www.fiveco.ch
info@fiveco.ch

Updates, Support and further Information available on the following web page:

http://www.fiveco.ch/bus-converter-products.html