



# **BA\_3722964-Millimar N1700 DLL Documentation\_EN.docx**

---

**Software N1700.DLL**

**Software version from V1.02-20**

14.05.2025

**Operating instructions**

3722964

**Mahr GmbH**

Reutlinger Str. 48, 73728 Esslingen

Tel.: +49 711 9312 600, Fax: +49 711 9312 756

mahr.es@mahr.de, [www.mahr.de](http://www.mahr.de)

# 1 Content

2	General information .....	4
3	Features .....	5
4	Supported modules.....	6
5	Installation using Windows.....	7
6	Files.....	8
7	Overview of the N1700.DLL .....	9
	Constants .....	9
	Constants for return values .....	9
	Constants for messages .....	9
	Enumeration types: Modul-Typ .....	10
	Enumeration types: port-Typ.....	10
	Enumeration types: Led statuses .....	10
	Enumeration types: Measured value data type .....	11
	Data structure DLL versions .....	11
	Data structure module .....	11
	Data structure channel.....	12
	Data structure channel channel data .....	12
	Data structure of configuration N 1702 VPP .....	12
	Function prototype - “Data-Callback”.....	13
	Function prototype – “Extended Data-Callback” .....	13
	Function prototype – “Message-Callback” .....	13
	Overview of all N 1700 functions .....	13
8	Description of the N 1700 functions .....	16
	N1700InitializeLibrary .....	16
	N1700InitializeLibraryNoAutoSrch.....	16
	N1700GetDevices .....	16
	N1700SetAutoSrch.....	16
	N1700FreeLibrary .....	16
	N1700Refresh.....	17
	N1700GetVersion.....	17
	N1700GetNumModules .....	17
	N1700GetNumChannels .....	17
	N1700GetVersion.....	17
	N1700GetChannel .....	18

N1700PollData .....	18
N1700RequestData.....	18
N1700RequestAllData.....	18
N1700StartContinuousRequestAllData.....	19
N1700StopContinuousRequestAllData.....	19
N1700StartContinuousRequestFootSwitch.....	19
N1700StopContinuousRequestFootSwitch .....	19
N1700SetData .....	19
N1700SetLED .....	19
N1700GetLED.....	20
N1700SetDecimals.....	20
N1700GetCalibration .....	20
N1700SetOffsetMM.....	20
N1700SetDigitsToMM.....	21
N1700SetGain .....	21
N1700GetFilter.....	21
N1700SetFilter .....	22
N1700GetPeType .....	22
N1700SetPeType.....	22
N1700GetCnfVPP.....	22
N1700SetCnfVPP.....	23
N1700DoResetVPP.....	23
N1700ActivatePhaseCorrVPP.....	23
N1700RegisterDataCallback .....	23
N1700UnRegisterDataCallback .....	24
N1700RegisterExtDataCallback.....	24
N1700UnRegisterExtDataCallback.....	24
N1700RegisterExtDataCallbackRaw .....	24
N1700UnRegisterExtDataCallbackRaw .....	24
N1700RegisterMsgCallback .....	25
N1700UnRegisterMsgCallback.....	25
N1700GetCustomerCalibration.....	25
N1700SetCustomerGain.....	25
N1700CustomerCalibrateChannelStart .....	26
N1700CustomerCalibrateChannelPos.....	26
N1700CustomerCalibrateChannelSave.....	26

N1700SetCustomerOffsetMM .....	26
N1700SetCustomerDigitsToMM .....	27
N1700ActivateCustomerCalibration .....	27
N1700ClearCustomerCalibration .....	27
9 Further explanation of modules .....	28
Parameter of the N 1702 VPP module .....	28
Process of customer calibration.....	31
10 Examples.....	33
11 Important Conditions to use the N1700.DLL .....	34

## **2 General information**

The N1700.DLL provides universal driver access to Millimar N 1700 modules. It supports the following

tasks:

- Detecting connected modules
- Reading individual measurements
- Starting/stopping continuous measurement for all modules
- Reading and setting the ports of digital I/O modules
- Reading the foot-operated switch
- Calibrating the modules
- Configuration of the modules

### 3 Features

- Universal driver access to Millimar N 1700 modules under Windows
- Connects modules with a total of up to 100 channels (max. 64 modules)
- Supports all features of Millimar N 1700 modules
- Different ways to retrieve measurements
  - o Via polling request
  - o Via measurement request in “Message Callback” function
  - o Via “Data Callback” function
- Reading the foot-operated switch using the “Message Callback” function when status of footoperated switch has changed
- Reading and setting the ports of digital I/O modules (N 1704 I/O)
- Specifying the configuration parameter of incremental modules (N 1702 VPP)
- Examples for C++, C# and Delphi

## 4 Supported modules

The table below demonstrates the supported N1700 modules according to the N1700.dll versions

N1700.dll version	Supported N1700 modules
V1.01-26	N 1702 USB, N 1702 M/T/U, N 1704 M/T/U, N 1704 I/O, N 1701 PM/PF, N 1702 M-HR
V1.02-10	N 1702 USB, N 1702 M/T/U, N 1704 M/T/U, N 1704 I/O, N 1701 PM/PF, N 1702 M-HR, N 1702 VPP

## 5 Installation using Windows

For the function to work in Windows, the **ftd2xx.DLL** file must be in the directory. This DLL is used to communicate with the FTDI chip. The **N1700.DLL** and **N1700\_64.DLL** DLLs must be copied to the directory of the application.



## 6 Files

The DLL exists in the **N1700.DLL** version for 32-bit Windows applications and in the **N1700\_64.DLL** version for 64-bit Windows applications.

The **N1700.H** header file is integrated for the use of the DLL in C++. The **N1700.INC** include file is integrated in Delphi.

The **MillimarN1700LibCSharp.dll** library, that is also available in the source text in sample project **MillimarN1700LibCSharp**, is required for C#.

The **ftd2xx.dll** file is required. This file is also stored in the appropriate 32- or 64-bit version in the respective directory of the sample programs.

## 7 Overview of the N1700.DLL

### Constants

```
#define MaxChannelCnt 4
#define NO_VAL -999999999
```

### Constants for return values

```
#define N1700_SUCCESS 0
#define N1700_FAILURE -1
#define N1700_TIMEOUT -2
#define N1700_INVALID_DEVNO -3
#define N1700_NO_MODULES -4
#define N1700_FILENOTEXISTS -5
#define N1700_WRONGFILEFORMAT -6
#define N1700_NOTYET_SUPPORTED -7
#define N1700_INVALID_CHANNELIDX -8
#define N1700_CONTINUOUS_ACTIVE -9
#define N1700_CALL_STILL_IN_ACTION -10
#define N1700_WRONG_MODULETYPE -11
#define N1700_FILEVARIANTNOTEXISTS -12
```

### Constants for messages

```
#define WM_USER 0x00000400
#define WM_N1700_Tick WM_USER + 2000 + 1
#define WM_N1700_ModuleCountChanged WM_USER + 2000 + 2
#define WM_N1700_ChannelCountChanged WM_USER + 2000 + 3
#define WM_N1700_NewMeasVal WM_USER + 2000 + 4
#define WM_N1700_SendDataCallbacks WM_USER + 2000 + 5
#define WM_N1700_MwProSek WM_USER + 2000 + 6
#define WM_N1700_Switch WM_USER + 2000 + 7
#define WM_N1700_Communication WM_USER + 2000 + 8
#define WM_N1700_FirmwareUpdateProgress WM_USER + 2000 + 9 // Param = Progress in 1/10%
#define WM_N1700_FirmwareUpdateError WM_USER + 2000 + 10
#define WM_N1700_Debug WM_USER + 2000 + 11
#define WM_N1700_ChannelMwProSek WM_USER + 2000 + 12
#define WM_N1700_ChannelParChanged WM_USER + 2000 + 13
#define WM_N1700_Error WM_USER + 2000 + 14
#define WM_N1700_ErrorFlashInfo WM_USER + 2000 + 15
#define WM_N1700_AutoReset WM_USER + 2000 + 16
#define WM_N1700_PhaseCorrTimeoutSecs WM_USER + 2000 + 17
#define WM_N1700_PhaseCorrValue WM_USER + 2000 + 18
#define WM_N1700_RefStat WM_USER + 2000 + 19 // 1: RefStat Ok, 0: RefStat Not Ok
```

## Enumeration types: Modul-Typ

```
enum tModuleType : byte
{
    mtUNDEF,
    mtPOWER,
    mtTERMINATION,
    mtN1701USB,
    mtN1702M,
    mtN1702T,
    mtN1702U,
    mtN1704M,
    mtN1704T,
    mtN1704U,
    mtN1704IO,
    mt1701PMXXXX,           // 20171207
    mt1701PM2500,           // 20171207
    mt1701PM5000,           // 20171207
    mt1701PM10000,          // 20171207
    mt1701PF25005000,       // 20171207
    mt1701PF25005000_4,     // 20171207
    mt1701PF10000,          // 20171207
    mtN1702M_HR,            // 20181210
    mtN1702VPP              // 20211116
};
```

## Enumeration types: port-Typ

```
enum tPortType : byte
{
    ptNone,
    ptAnalog,
    ptDigital,
    ptIncr
};
```

## Enumeration types: Led statuses

```
enum tLedState : byte
{
    lsOff,
    lsOn,
    lsBlocked,
    lsUnblocked
};
```

## Enumeration types: Measured value data type

```
enum tDataValueType : byte
{
    dvtDigital,           // Datatype for N1704IO
    dvtPosition,          // Datatype for all modules (except N1704IO)
    dvtVelocity,          // Datatype for N1702VPP Velocity
    dvtTurn,              // Datatype for N1702VPP Multiturn
    dvtPositionRaw,       // Datatype for Rawvalues (all modules except N1701USB, N1704IO)
    dvtVelocityRaw        // Datatype for N1702VPP Velocity Rawvalues
};
```

## Data structure DLL versions

```
struct N1700version {
    struct {
        int major;
        int minor;
        int micro;
        int nano;
    }N1700lib;
    struct {
        int major;
        int minor;
        int micro;
        int nano;
    }FTDILib;
}; // N1700VERSION, *N1700VERSION;
```

## Data structure module

```
typedef struct sN1700_Module { //Call of function N1700GetModule fills this struct
    UI32 ModuleIdx;           // Module Id, sequence number
    byte sFtDescription[40];   // For internal use
    byte sFtSerial[12];        // For internal use
    tModuleType ModuleType;    // Type of Module
    byte sModuleType[24];      // Type of Module as string
    byte sDescription[24];     // Name of Module as string, same as sModuleType
    byte sIdentNo[8];          // Ident Number of Module
    byte sSerialNo[9];         // Serial Number of Module
    byte sFirmwareVersion[10]; // Firmware Version number of Module
    byte ChannelCount;         // Channel count of Module
    byte PowerModuleNeeded;    // If TRUE, Power Module needed in front of this Module
    short int PowerConsumption_mA; // Power Consumption of Module, negative if
                                   // Consumption, positiv if Supply (USB-Module, Power
                                   // Module)
    UI32 ChannelIdxArray[4];   // The Ids of the Channels included
};
```

Pointers to variables of this data structure are transferred to the N1700GetModule function.

## Data structure channel

```

struct sN1700_Channel {      // Call of function N1700GetChannel fills this struct
    UI32 ChannelIdx;         // The Id of the Channel
    UI32 ParentModuleIdx;    // The Id of the Parent Module for accessing the Module-Parameters
    tPortType tPortType;     // Type of Channel-Port (ptAnalog, ptDigital, ptIncremental)
    byte PortInCount;        // Count of Input Ports
    byte PortOutCount;       // Count of Output Ports
    int Decimals;            // Count of decimals for PortType ptAnalog and ptIncremental. Can be
                             // Changed with Call of N1700SetDecimal
    UI32 DigFilter;          // Averaging filter constant
    byte CustomerCalibActive;
    byte CustomerCalibrated;
    byte FactoryCalibrated;
};      // N1700_Channel, *PN1700_Channel

```

Pointers to variables of this data structure are transferred to the N1700GetModule function.

## Data structure channel channel data

```

struct sN1700_ChannelExtData {    // for use with ExtDataCallback
    UI32 ChannelIdx;
    tDataValueType ValueType;     // type of data
    byte Reserve1[3];              // for future use
    double dValue;                 // data
    byte ReferenceActive;          // N 1702 VPP flag
    byte Referenced;              // N 1702 VPP flag
    byte Reserve2[14];            // for future use
};

```

## Data structure of configuration N 1702 VPP

```

struct sN1700_CnfVPP {
    bool PhasenCorrOn;             // activates the phase shift correction
    bool PhasenCorrOk;            // phase shift correction used in past
    byte PhaseCorrDeg10Value;      // phase shift value in unit 0.1 degrees, ReadOnly
    bool RotaryNotLinear;         // defines the encoder type
    byte IPolFaktIdx;              // Interpolation factor
    bool PosAndVel;                // Position or Postion + Velocity output
    bool RefActive;                // enables Reference point processing
    bool RefStatOk;                // indicates status reference point
    bool MultiTurn;                // enables the multi cycle counter
    bool FilterOn;                 // enables the analog filter
    byte FilterFreqIdx;            // Analog filter cutoff frequency
    byte Reserve1[1];              // for future use
    UI32 PerLenOrIncPR;            // Period length or increments
    UI32 DistanceRefMarkers;       // Distance between reference points or Reference points per
                                   // revolution
    byte Reserve2[16];             // for future use
};

```

## Function prototype - “Data-Callback”

```
typedef int(__stdcall *F_N1700DataCallback)(
    int numChannels,          // Count of Channels, from which will Data is received
    int *pChannelIdxArray,    // Array with Channel-Ids, from which will Data is send
    double* pData,           // pointer to float array with measuring data
    void* pContext);
// Additional explanation of *pdata:
// if Channel is an N 1704 I/O, the float value has to be converted to uint32,
// the low 2 bytes are the digital outputs (Bytes 0x0001 is Port 1; Bytes 0x0002 is Port 2 ...
// the high 2 bytes are the digital inputs (Bytes 0x0001 is Port 1; Bytes 0x0002 is Port 2 ...
```

## Function prototype – “Extended Data-Callback”

```
typedef int(__stdcall *F_N1700ExtDataCallback)(
    int numChannels,          // Count of Channels, from which will Data is received
    sN1700_ChannelExtData* pData, // Pointer to array of sN1700_ChannelExtData
    void* pContext);
```

## Function prototype – “Message-Callback”

```
typedef int(__stdcall *F_N1700MsgCallback)(
    int msg,
    UI32 Channel,
    int param);
```

## Overview of all N 1700 functions

An overview of all available functions is provided in the following. The individual functions are explained individually after this chapter.

### General functions

```
typedef int(__stdcall *F_N1700InitializeLibrary)(BOOL Console, UI32 *NumModules, UI32
*NumChannels, I32 Par);
typedef int(__stdcall *F_N1700InitializeLibraryNoAutoSrch)(BOOL Console, I32 Par);
typedef int(__stdcall *F_N1700GetDevices)(UI32 *NumModules, UI32 *NumChannels);
typedef int(__stdcall *F_N1700SetAutoSrch)(BOOL AutoSrch);
typedef int(__stdcall *F_N1700FreeLibrary)(void);
typedef int(__stdcall *F_N1700Refresh)(UI32 *NumModules, UI32 *NumChannels);
```

### General functions concerning modules and module parameters

```
typedef int(__stdcall *F_N1700GetVersion)(N1700version *Version);
typedef int(__stdcall *F_N1700GetNumModules)(void);
typedef int(__stdcall *F_N1700GetNumChannels)(void);
typedef int(__stdcall *F_N1700GetModule)(UI32 ModuleIdx, sN1700_Module* Module);
// sN1700_Module
typedef int(__stdcall *F_N1700GetChannel)(UI32 ChannelIdx, sN1700_Channel* Channel);
// sN1700_Channel;
typedef int(__stdcall *F_N1700SetLED)(UI32 channelIdx, tLedState state);
```

```
typedef int(__stdcall *F_N1700GetLED)(UI32 channelIdx, tLedState* state);
typedef int(__stdcall *F_N1700SetDecimals)(UI32 channelIdx, int decimals);
typedef int(__stdcall *F_N1700GetFilter)(UI32 channelIdx, int* Filter);
typedef int(__stdcall *F_N1700SetFilter)(UI32 channelIdx, int Filter);
```

### Functions concerning data acquisition

```
typedef int(__stdcall *F_N1700PollData)(UI32 ChannelIdx, double* Data);
// Pointer to array of ChannelId
typedef int(__stdcall *F_N1700RequestData)(int numChannels, UI32* pchannelIdxArray, int par);
// Pointer to array of ChannelId
typedef int(__stdcall *F_N1700RequestAllData)(int par);
typedef int(__stdcall *F_N1700StartContinuousRequestAllData)(UI32 interval, int par);
typedef int(__stdcall *F_N1700StopContinuousRequestAllData)(void);
typedef int(__stdcall *F_N1700StartContinuousRequestFootSwitch)(void);
typedef int(__stdcall *F_N1700StopContinuousRequestFootSwitch)(void);
```

### Functions concerning factory calibration (do not use!)

```
typedef int(__stdcall *F_N1700GetCalibration)(UI32 channelIdx, float* OffsetMM, float*
digitsToMM, float* gain);
typedef int(__stdcall *F_N1700SetGain)(UI32 channelIdx, float gain);
typedef int(__stdcall *F_N1700SetOffsetMM)(UI32 channelIdx, float offsetMM);
typedef int(__stdcall *F_N1700SetDigitsToMM)(UI32 channelIdx, float digitsToMM);
```

### Function concerning the N 1702 I/O module

```
typedef int(__stdcall *F_N1700SetData)(UI32 channelIdx, UI32 data);
```

### Functions concerning N 1702 PM/PF modules

```
typedef int(__stdcall *F_N1700GetPeType)(UI32 channelIdx, int* PeType);
typedef int(__stdcall *F_N1700SetPeType)(UI32 channelIdx, int PeType);
```

### Functions concerning N 1702 VPP module

```
typedef int(__stdcall *F_N1700GetCnfVPP)(UI32 channelIdx, bool ReadNew, sN1700_CnfVPP*
CnfVPP);
typedef int(__stdcall *F_N1700SetCnfVPP)(UI32 channelIdx, sN1700_CnfVPP CnfVPP);
typedef int(__stdcall *F_N1700DoResetVPP)(UI32 channelIdx);
typedef int(__stdcall *F_N1700ActivatePhaseCorrVPP)(UI32 channelIdx, byte* PhaseCorr);
```

### Callback functions

```
typedef int(__stdcall *F_N1700RegisterDataCallback)(F_N1700DataCallback pCallback, int
numChannels, int *pChannelIdxArray, void *pContext);
typedef int(__stdcall *F_N1700UnregisterDataCallback)(F_N1700DataCallback pCallback);
typedef int(__stdcall *F_N1700RegisterExtDataCallback)(F_N1700ExtDataCallback pCallback, int
numChannels, int *pChannelIdxArray, void *pContext);
typedef int(__stdcall *F_N1700UnregisterExtDataCallback)(F_N1700ExtDataCallback pCallback);
typedef int(__stdcall *F_N1700RegisterExtDataCallbackRaw)(F_N1700ExtDataCallback pCallback,
int numChannels, int *pChannelIdxArray, void *pContext);
```

```
typedef int(__stdcall *F_N1700UnregisterExtDataCallbackRaw)(F_N1700ExtDataCallback  
pCallback);  
typedef int(__stdcall *F_N1700RegisterMsgCallback)(F_N1700MsgCallback pCallback);  
typedef int(__stdcall *F_N1700UnregisterMsgCallback)(void);
```

### **Functions concerning customer calibration**

```
typedef int(__stdcall *F_N1700GetCustomerCalibration)(UI32 channelIdx, float* offsetMM, float*  
digitsToMM, float* gain, bool* isActive);  
typedef int(__stdcall *F_N1700CustomerCalibrateChannelStart)(UI32 channelIdx);  
typedef int(__stdcall *F_N1700CustomerCalibrateChannelPos)(int calPos, double calValMM, int*  
calValDigits);  
typedef int(__stdcall *F_N1700CustomerCalibrateChannelSave)(void);  
typedef int(__stdcall *F_N1700SetCustomerGain)(UI32 channelIdx, float gain);  
typedef int(__stdcall *F_N1700SetCustomerOffsetMM)(UI32 channelIdx, float offsetMM);  
typedef int(__stdcall *F_N1700SetCustomerDigitsToMM)(UI32 channelIdx, float digitsToMM);  
typedef int(__stdcall *F_N1700ActivateCustomerCalibration)(UI32 channelIdx, bool activate);  
typedef int(__stdcall *F_N1700ClearCustomerCalibration)(UI32 channelIdx);
```



## 8 Description of the N 1700 functions

### N1700InitializeLibrary

The first Call of DLL initializes Library and determines the connection configuration.

```
typedef int(__stdcall *F_N1700InitializeLibrary)(BOOL Console, UI32 *NumModules, UI32 *NumChannels, I32 Par);
```

**In Parameter:**

Console: TRUE, if used in Console-Application. FALSE, if Used in Windows-Forms-Application

**Out Parameters:**

NumModules: Count of connected Modules

NumChannels: Count of all channels in connected Modules

### N1700InitializeLibraryNoAutoSrch

Initialize Library for use in applications that connect to other FTDI-Devices.

```
typedef int(__stdcall *F_N1700InitializeLibraryNoAutoSrch)(BOOL Console, I32 Par);
```

**In Parameter:**

Console: TRUE, if used in Console-Application. FALSE, if Used in Windows-Forms-Application

### N1700GetDevices

Get DeviceCount if FTDI-Count changed when initialized with **N1700InitializeLibraryNoAutoSrch**

```
typedef int(__stdcall *F_N1700GetDevices)(UI32 *NumModules, UI32 *NumChannels);
```

**Out Parameters:**

NumModules: Count of connected Modules

NumChannels: Count of all channels in connected Modules

### N1700SetAutoSrch

Activate or Deactivate **AutoSrch** N1700 Devices .

```
typedef int(__stdcall *F_N1700SetAutoSrch)(BOOL AutoSrch);
```

**In Parameters:**

AutoSrch: true or false

### N1700FreeLibrary

Closing the DLL-Connection

```
typedef int(__stdcall *F_N1700FreeLibrary)(void);
```

## N1700Refresh

Determine connected configuration again

```
typedef int(__stdcall *F_N1700Refresh)(UI32 *NumModules, UI32 *NumChannels);
```

### Out Parameters:

NumModules: Count of connected Modules

NumChannels: Count of all channels in connected Modules

## N1700GetVersion

```
typedef int(__stdcall *F_N1700GetVersion)(N1700version *Version);
```

### Out Parameter:

Version: Pointer to Version of DLLs in struct N1700version

## N1700GetNumModules

Determine count of connected Modules

```
typedef int(__stdcall *F_N1700GetNumModules)(void);
```

### Out Parameter:

Count of connected Modules

## N1700GetNumChannels

Determine count of connected Channels

```
typedef int(__stdcall *F_N1700GetNumChannels)(void);
```

### Return Parameter:

Count of all channels in connected Modules

## N1700GetModule

Read Information of Module

```
typedef int(__stdcall *F_N1700GetModule)(UI32 ModuleIdx, sN1700_Module* Module); //  
sN1700_Module
```

### In Parameter:

ModuleIdx: Module Id, whose Information will be returned (0 to ModuleCount-1)

### Out Parameter:

Module: Pointer to struct sN1700\_Module

## N1700GetChannel

Read Information of Channel

```
typedef int(__stdcall *F_N1700GetChannel)(UI32 ChannelIdx, sN1700_Channel* Channel);
// sN1700_Channel
```

### In Parameter:

ChannelIdx: Id of Channel, from which Information will be returned (0..ChannelCount-1)

### Out Parameter:

Channel: Pointer to struct sN1700\_Channel

## N1700PollData

Read Data-Value from Channel. If reading N 1702 VPP channels, only the singleturn data is returned (position or degrees, dependent of configuration). For fast measurement, it is highly recommended to call the Function **N1700StartContinuousRequestAllData** before.

```
typedef int(__stdcall *F_N1700PollData)(UI32 ChannelIdx, double* Data);
// Pointer to array of ChannelId
```

### In Parameters:

ChannelIdx: Id of Channel, from which Information will be returned (0..ChannelCount-1)

### Out Parameter:

Data: Pointer to read measuring Value

## N1700RequestData

Request Data of selected Channels

```
typedef int(__stdcall *F_N1700RequestData)(int numChannels, UI32* pchannelIdxArray, int par);
// Pointer to array of ChannelId
```

### In Parameters:

numChannels: Count of Channels, from which data will be requested

pChNoArray: Array with Channel-Ids, , from which will data be requested

par: for internal use

## N1700RequestAllData

Request Data of all Channels

```
typedef int(__stdcall *F_N1700RequestAllData)(int par);
```

### In Parameters:

par: for internal use

**N1700StartContinuousRequestAllData**

Start Continuous Request of Data for all Channels

```
typedef int(__stdcall *F_N1700StartContinuousRequestAllData)(UI32 interval, int par);
```

**In Parameter:**

interval: not supported yet (for future use), Data will be received as fast as possible

par: for internal use

**N1700StopContinuousRequestAllData**

Stop Continuous Request of Data for all Channels

```
typedef int(__stdcall *F_N1700StopContinuousRequestAllData)(void);
```

**N1700StartContinuousRequestFootSwitch**

Start Continuous Request of Foot Switch

```
typedef int(__stdcall *F_N1700StartContinuousRequestFootSwitch)(void);
```

**N1700StopContinuousRequestFootSwitch**

Stops Continuous Request of Foot Switch

```
typedef int(__stdcall *F_N1700StopContinuousRequestFootSwitch) (void);
```

**N1700SetData**

Sets the ports of Channel of Module "N 1704 I/O"-Module

```
typedef int(__stdcall *F_N1700SetData)(UI32 channelIdx, UI32 data);
```

**In Parameters:**

ChannelIdx: Id of Channel. Must be the Channel of a "N 1704 I/O"-Module

data: state of Ports bitwise:

- Port Out 01: (0x0001) (0b0000 0000 0000 0001)
- Port Out 02: (0x0002) (0b0000 0000 0000 0010)
- Port Out 03: (0x0003) (0b0000 0000 0000 0011)
- Port Out 04: (0x0004) (0b0000 0000 0000 0100)

**N1700SetLED**

Sets the LED state of Channel

```
typedef int(__stdcall *F_N1700SetLED)(UI32 channelIdx, tLedState state);
```

**In Parameters:**

ChannelIdx: Id of Channel

state: LED State: 0 = OFF, 1 = ON, 2 = BLOCKED, 3 = UNBLOCKED

## N1700GetLED

Gets the LED state of Channel

```
typedef int(__stdcall *F_N1700GetLED)(UI32 channelIdx, tLedState* state);
```

### In Parameters:

ChannelIdx: Id of Channel

Out Parameters:

state: LED State: 0 = OFF, 1 = ON, 2 = BLOCKED, 3 = UNBLOCKED

## N1700SetDecimals

Sets the amount of decimals of measuring value. Only for optional use. The amount of decimals will be saved in struct **sN1700\_Channel**

```
typedef int(__stdcall *F_N1700SetDecimals)(UI32 channelIdx, int decimals);
```

### In Parameters:

ChannelIdx: Id of Channel

decimals: Count of decimals of measuring value

## N1700GetCalibration

Get Calibration Parameters

```
typedef int(__stdcall *F_N1700GetCalibration)(UI32 channelIdx, float* OffsetMM, float* digitsToMM, float* gain);
```

### In Parameters:

ChannelIdx: Id of Channel

### Out Parameters:

OffsetMM: Offset value in millimeter

digitsToMM: Amount of digits (of rawvalue) for the length of one millimeter

gain: Gain value

The measuring values will be calculated as follows. The Rawvalue is an internal ADC value of the inductive modules.

$$Measuring\ Value = \left( \frac{Rawvalue}{DigitsToMM} - OffsetMM \right) \cdot Gain$$

## N1700SetOffsetMM

**For internal use only! Do not use! Using this function will decalibrate the module!**

Set Calibration Parameter offsetMM

```
typedef int(__stdcall *F_N1700SetOffsetMM)(UI32 channelIdx, float offsetMM);
```

### In Parameters:

ChannelIdx: Id of Channel

OffsetMM: Offset value in unit millimeter

The measuring values will be calculated as follows. The Rawvalue is an internal ADC value of the inductive modules.

$$Measuring\ Value = \left( \frac{Rawvalue}{DigitsToMM} - OffsetMM \right) \cdot Gain$$

## N1700SetDigitsToMM

**For internal use only! Do not use! Using this function will decalibrate the module!**

Set Calibration Parameter digitsToMM

```
typedef int(__stdcall *F_N1700SetDigitsToMM)(UI32 channelIdx, float digitsToMM);
```

### In Parameters:

ChannelIdx: Id of Channel

digitsToMM: Amount of digits (of rawvalue) for the length of one millimeter

The measuring values will be calculated as follows. The Rawvalue is an internal ADC value of the inductive modules.

$$Measuring\ Value = \left( \frac{Rawvalue}{DigitsToMM} - OffsetMM \right) \cdot Gain$$

## N1700SetGain

**For internal use only! Do not use! Using this function will decalibrate the module!**

Set Calibration Parameter gain

```
typedef int(__stdcall *F_N1700SetGain)(UI32 channelIdx, float gain);
```

### In Parameters:

ChannelIdx: Id of Channel

Gain: Gain value

$$Measuring\ Value = \left( \frac{Rawvalue}{DigitsToMM} - OffsetMM \right) \cdot Gain$$

## N1700GetFilter

Gets the configured digital averaging filter of the selected channel. Default value is 5. Be aware, that this value could have been changed via MillimarN1700.exe software in the past.

```
typedef int(__stdcall *F_N1700GetFilter)(UI32 channelIdx, int* Filter);
```

### In Parameters:

ChannelIdx: Id of Channel

### Out Parameters:

Filter: Length of the digital averaging filter.

0 => averaging filter deactivated

1 => averaging of 2 values

2 => averaging of 4 values

3 => averaging of 8 values

4 => averaging of 16 values

5 => averaging of 32 values

6 => averaging of 64 values

## N1700SetFilter

Gets the configured digital averaging filter of the selected channel.

```
typedef int(__stdcall *F_N1700SetFilter)(UI32 channelIdx, int Filter);
```

### In Parameters:

ChannelIdx: Id of Channel

Filter: Length of the digital averaging filter. For valid values, see **N1700GetFilter**.

## N1700GetPeType

**Only for internal use! Do Not use!**

Gets the type of the PE-module.

```
typedef int(__stdcall *F_N1700GetPeType)(UI32 channelIdx, int* PeType);
```

### In Parameters:

ChannelIdx: Id of Channel

### Out Parameters:

PeType: Type of Pe-module

## N1700SetPeType

**Only for internal use! Do Not use! Using this function will lead to wrong measuring values!**

Sets the type of the PE-module.

### In Parameters:

ChannelIdx: Id of Channel

PeType: Type of Pe-module

## N1700GetCnfVPP

Gets the configuration data of the N 1700 VPP module

```
typedef int(__stdcall *F_N1700GetCnfVPP)(UI32 channelIdx, bool ReadNew, sN1700_CnfVPP* CnfVPP);
```

### In Parameters:

ChannelIdx: Id of Channel

ReadNew: When True, the internally stored configuration data of the module will be read again. It will be automatically read while first connection of the module.

### Out Parameters:

CnfVPP: Configuration structure of N 1702 VPP module

## N1700SetCnfVPP

Sets the configuration data of the N 1700 VPP module.

```
typedef int(__stdcall *F_N1700SetCnfVPP)(UI32 channelIdx, sN1700_CnfVPP CnfVPP);
```

### In Parameters:

ChannelIdx: Id of Channel

CnfVPP: Configuration structure of N 1702 VPP module

## N1700DoResetVPP

Reset of the fault flags of N1702VPP. Additionally the measuring values (singleturn and multiturn) will be set to zero. This function can be used for relative measurement.

```
typedef int(__stdcall *F_N1700DoResetVPP)(UI32 channelIdx);
```

### In Parameters:

ChannelIdx: Id of Channel

## N1700ActivatePhaseCorrVPP

**This function is only for internal use. Do not use! Not supported yet!**

Starts the internal phase shift correction process

```
typedef int(__stdcall *F_N1700ActivatePhaseCorrVPP)(UI32 channelIdx, byte* PhaseCorr);
```

### In Parameters:

ChannelIdx: Id of Channel

### Out Parameters:

PhaseCorr: Phase shift correction value of N 1702 VPP module

## N1700RegisterDataCallback

Register a callback that will be called every time new values arrives

```
typedef int(__stdcall *F_N1700RegisterDataCallback)(F_N1700DataCallback pCallback, int numChannels, int *pChannelIdxArray, void *pContext);
```

### In Parameters:

pCallback: Callback function

numChannels: Count of Channels, from which will Data be requested

pChannelIdxArray: Array with Channel-Ids, from which will Data be requested



## N1700UnRegisterDataCallback

Unregister a callback function

```
typedef int(__stdcall *F_N1700UnregisterDataCallback)(F_N1700DataCallback pCallback);
```

### In Parameters:

pCallback: Callback function

## N1700RegisterExtDataCallback

Register a callback that will be called every time new values arrives

```
typedef int(__stdcall *F_N1700RegisterExtDataCallback)(F_N1700DataCallback pExtCallback, int numChannels, int *pChannelIdxArray, void *pContext);
```

### In Parameters:

pExtCallback: Extended Callback function

numChannels: Count of Channels, from which will Data be requested

pChannelIdxArray: Array with Channel-Ids, from which will Data be requested

## N1700UnRegisterExtDataCallback

Unregister a callback function

```
typedef int(__stdcall *F_N1700UnregisterExtDataCallback)(F_N1700ExtDataCallback pExtCallback);
```

### In Parameters:

pExtCallback: Extended Callback function

## N1700RegisterExtDataCallbackRaw

Register a callback that will be called every time new raw values arrives

```
typedef int(__stdcall *F_N1700RegisterExtDataCallbackRaw)(F_N1700DataExtCallback pExtCallback, int numChannels, int *pChannelIdxArray, void *pContext);
```

### In Parameters:

pExtCallback: Extended Callback function

numChannels: Count of Channels, from which will Data be requested

pChannelIdxArray: Array with Channel-Ids, from which will Data be requested

## N1700UnRegisterExtDataCallbackRaw

Unregister a callback function

```
typedef int(__stdcall *F_N1700UnregisterExtDataCallbackRaw)(F_N1700ExtDataCallback pExtCallback);
```

**In Parameters:**

pExtCallback: Extended Callback function

**N1700RegisterMsgCallback**

Register a callback for receiving Messages from DLL

```
typedef int(__stdcall *F_N1700RegisterMsgCallback)(F_N1700MsgCallback pCallback);
```

**In Parameters:**

pCallback: Callback function

**N1700UnRegisterMsgCallback**

Unregister the Message callback function

```
typedef int(__stdcall *F_N1700UnregisterMsgCallback)(void);
```

**N1700GetCustomerCalibration**

Get Calibration Parameters

```
typedef int(__stdcall *F_N1700GetCustomerCalibration)(UI32 channelIdx, float* OffsetMM, float* digitsToMM, float* gain, bool* isActive);
```

**In Parameters:**

ChannelIdx: Id of Channel

**Out Parameters:**

OffsetMM, digitsToMM, gain

isActive: TRUE, if Customer calibration is activated

$$Measuring\ Value = \left( \frac{Rawvalue}{DigitsToMM} - OffsetMM \right) \cdot Gain$$

**N1700SetCustomerGain**

Sets Calibration Parameter gain

```
typedef int(__stdcall *F_N1700SetCustomerGain)(UI32 channelIdx, float gain);
```

**In Parameters:**

ChannelIdx: Id of Channel

Gain: Parameter gain

$$Measuring\ Value = \left( \frac{Rawvalue}{DigitsToMM} - OffsetMM \right) \cdot Gain$$

## N1700CustomerCalibrateChannelStart

Starts the Customer Calibration of a channel

```
typedef int(__stdcall *F_N1700CustomerCalibrateChannelStart)(UI32 channelIdx);
```

### In Parameters:

ChannelIdx: Id of Channel

## N1700CustomerCalibrateChannelPos

**For internal use only! Do not use! Using this function will decalibrate the module!**

Execute Customer Calibration of Position of selected Calibration-Channel (See N1700CustomerCalibrateChannelStart)

```
typedef int(__stdcall *F_N1700CustomerCalibrateChannelPos)(int calPos, double calValMM, int* calValDigits);
```

### In Parameters:

calPos: Id of Calibration-Position: -1, 0, or +1

calValMM: The value of position in mm to Measure

### Out Parameters:

calValDigits: The measured value of position in digits

## N1700CustomerCalibrateChannelSave

**Caution! Customer calibration values will be stored inside the module! Existing values will be overwritten!**

Save The Customer Calibration of selected Calibration-Channel (See N1700CustomerCalibrateChannelStart)

```
typedef int(__stdcall *F_N1700CustomerCalibrateChannelSave)(void);
```

## N1700SetCustomerOffsetMM

Set Calibration Parameter offsetMM for customer calibration

```
typedef int(__stdcall *F_N1700SetCustomerOffsetMM)(UI32 channelIdx, float offsetMM);
```

### In Parameters:

ChannelIdx: Id of Channel

OffsetMM: Offset value in unit millimeter

The measuring values will be calculated as follows. The Rawvalue is an internal ADC value of the inductive modules.

$$Measuring\ Value = \left( \frac{Rawvalue}{DigitsToMM} - OffsetMM \right) \cdot Gain$$

## N1700SetCustomerDigitsToMM

Set Calibration Parameter digitsToMM for customer calibration

```
typedef int(__stdcall *F_N1700SetCustomerDigitsToMM)(UI32 channelIdx, float digitsToMM);
```

### In Parameters:

ChannelIdx: Id of Channel

digitsToMM: Amount of digits (of rawvalue) for the length of one millimeter

The measuring values will be calculated as follows. The Rawvalue is an internal ADC value of the inductive modules.

$$Measuring\ Value = \left( \frac{Rawvalue}{DigitsToMM} - OffsetMM \right) \cdot Gain$$

## N1700ActivateCustomerCalibration

Select between customer and factory calibration. Customer calibration (activate = TRUE) or factory calibration (activate = FALSE)

```
typedef int(__stdcall *F_N1700ActivateCustomerCalibration)(UI32 channelIdx, bool activate);
```

### In Parameters:

ChannelIdx: Id of Channel

Activate: TRUE: Customer calibration

FALSE: Factory calibration

## N1700ClearCustomerCalibration

**Caution! Clearing the customer calibration cannot be undone!**

Clears/Deletes the Customer Calibration

```
typedef int(__stdcall *F_N1700ClearCustomerCalibration)(UI32 channelIdx);
```

### In Parameters:

ChannelIdx: Id of Channel

## 9 Further explanation of modules

### Parameter of the N 1702 VPP module

The channels of the N 1702 VPP module are configured via the **sN1700\_CnfVPP** data structure and the

DLL functions **F\_N1700GetCnfVPP** and **F\_N1700SetCnfVPP**.

```
struct sN1700_CnfVPP {
    bool PhasenCorrOn;
    bool PhasenCorrOk;
    byte PhaseCorrDeg10Value;
    bool RotaryNotLinear;
    byte IPolFaktIdx;
    bool PosAndVel;
    bool RefActive;
    bool RefStatOk;
    bool MultiTurn;
    bool FilterOn;
    byte FilterFreqIdx;
    byte Reserve1[2];
    UI32 PerLenOrIncPR;
    UI32 DistanceRefMarkers;
    byte Reserve2[16];
};
```

The N 1702 VPP module must be configured before use. The structure parameters must be initialized for this purpose. This can either be completed directly by assigning the structure values or alternatively via the configuration menu. The following figure shows an example of a configuration menu (in this case, the configuration menu of the setup software MillimarN1700.exe). This menu enables the setting of the parameters of the N 1702 VPP module.

Setup N 1702 VSS Channel 2

☐ Linear Encoder  
☒ Rotary Encoder

☐ Up to one Reference Point  
☒ Multiple Reference Points

☐ Position  
☒ Position + Velocity

☒ Reference Point-Processing  
☐ Phase Shift Correction

**Start Phase Shift Correction**  
Phase Shift: 0,0 °

**Increments per Revolution**  
+3600

**Reference Points per Revolution**  
+1

**Analog Filter Cutoff Frequency (-1dB)**  
450 kHz

**Interpolation Factor**  
256

Cancel Save

The displayed menu can be used as a guide for your own settings. The parameters of the sN1700\_CnfVPP structure must be assigned correctly. The configuration parameters are explained below using the configuration menu illustrated above as an example.

### Linear or rotary encoder

The calculated measurement is dependent on this setting. It must be specified whether the sensor operates linearly or rotationally. The measurement is specified in degrees or millimeters depending on the setting.

Variable in sN1700\_CnfVPP: **bool RotaryNotLinear**

0 = Linear Encoder (for example: scale or probe; measurement in millimeters)

1 = Rotary Encoder (encoder; measurement in degrees)

### Period length or increments per turn

The parameter is dependent on the **RotaryNotLinear** variable. If **RotaryNotLinear** = 0, the electric period length must be specified in micrometers. If **RotaryNotLinear** = 1, the number of increments per turn of the connected rotational measuring system must be transferred.

Variable in sN1700\_CnfVPP: **UI32 PerLenOrIncPR**

value range: 0 bis 65535

### Distance between reference points or reference points per turn

The parameter is dependent on the **RotaryNotLinear** variable. Parameter is only required if the MultiTurn value is activated (**MultiTurn** parameter = 1). If **RotaryNotLinear** = 0, the distance between the reference points must be specified in micrometers. If **RotaryNotLinear** = 1, the number of reference points per turn must be specified.

Variable in sN1700\_CnfVPP: **UI32 DistanceRefMarkers**

value range: 0 bis 65535

### Interpolation factor

Specification of the interpolation factor. This parameter determines the relationship between the electrical period length and interpolation factor.

$$Resolution = \frac{electrical\ period\ length}{interpolation\ factor}$$

Variable in sN1700\_CnfVPP: **byte IPolFaktIdx**

Valid values: IPolFaktIdx = 0 => Interpolation factor = 256

IPolFaktIdx = 1 => Interpolation factor = 128

IPolFaktIdx = 2 => Interpolation factor = 64

IPolFaktIdx = 3 => Interpolation factor = 32

In the configuration menu above, the interpolation factor can be selected via a dropdown menu, for example.

## MultiTurn or SingleTurn

There is an option to calculate a MultiTurn value (number of counted reference points) in addition to a SingleTurn value (position/angle). It must be noted that only 22 bit will be used if the MultiTurn value is activated. The valid value ranges are detailed in the separate documentation for the N 1702 VPP module. Activation of the MultiTurn value requires the **RefActive** variable to also be set.

Variable in sN1700\_CnfVPP: **bool MultiTurn**

MultiTurn = 0 => Only the SingleTurn value is calculated

MultiTurn = 1 => SingleTurn value + MultiTurn value are calculated

## Reference point processing

This parameter can be used to determine whether the sensor reference point should be used to determine the measurement. If the reference point is passed, this will reset the SingleTurn value. If the reference point is passed and the MultiTurn value is activated, the MultiTurn value is incremented with the correct sign.

Variable in sN1700\_CnfVPP: **bool RefActive**

RefActive = 0 => Reference point is not evaluated

RefActive = 1 => Reference point is evaluated

## Additional calculation of the velocity

The (angle) velocity can be calculated in addition to the position. Dependent on the **RotaryNotLinear** parameter, either the velocity or the angle velocity is calculated. The time basis for calculating the (angle) velocity is 1 millisecond.

Variable in sN1700\_CnfVPP: **bool PosAndVel**

PosAndVel = 0 => Only position/angle

PosAndVel = 1 => Position + (angle) velocity

In the configuration menu illustrated above, buttons with the designations "Position" and "Position + Velocity" are used for this setting.

## Measurement filtering

On the N 1702 VPP module there is an option for the analog filtering of the sin/cos input signals. This filtering can be activated via the **FilterOn** parameter. The **TpFreqIdx** parameter is used to specify the cutoff frequency of the low pass filter. The frequencies 450 kHz, 200 kHz, 75 kHz and 10 kHz can be selected.

Variables in sN1700\_CnfVPP: **bool FilterOn; byte TpFreqIdx**

FilterOn = 0 => Analoges Tiefpassfilter wird deaktiviert

FilterOn = 1 => Analoges Tiefpassfilter wird aktiviert

TpFreqIdx = 0 => Grenzfrequenz 450kHz

TpFreqIdx = 1 => Grenzfrequenz 200kHz

TpFreqIdx = 2 => Grenzfrequenz 75kHz

TpFreqIdx = 3 => Grenzfrequenz 10kHz

Im oben dargestellten Konfigurationsmenü kann die analoge Filterung über ein Dropdownmenü konfiguriert werden. Das deaktivieren des Filters ebenfalls über das Dropdownmenü und den Wert “-“ möglich

### Correction of the phase shift of sin and cos

**The following parameters are only intended for internal test purposes and must not be assigned! The functions following the variables have not been released yet and can thus not be used!**

Variables in sN1700\_CnfVPP: **bool PhasenCorrOn; bool PhasenCorrOk; byte PhaseCorrDeg10Value**

PhasenCorrOn = 0 => Phase shift correction is deactivated

PhasenCorrOn = 1 => Phase shift correction activated

PhasenCorrOk = 0 => Phase shift correction has never been executed

PhasenCorrOk = 1 => Phase shift correction had been executed in the past

PhaseCorrDeg10Value => Phase shift value (value range -4,9° to 4,9°)

### RefStatOk parameter

The **RefStatOk** parameter specifies whether referencing has already been completed at the reference point. Referencing is completed the first time a reference point is passed. The **RefStatOk** parameter is updated every time a measurement is retrieved from the N 1702 VPP module. The **RefStatOk** parameter is only updated if the RefActive parameter is set in the structure.

Variable in sN1700\_CnfVPP: **bool RefStatOk**

RefStatOk = 0 => Referencing not completed (yet)

RefStatOk = 1 => Referencing completed at reference point

### Process of customer calibration

Modules with connection of inductive probes (N 1702 M, N 1702 T, N 1702 U, N 1702 M-HR, N 1704 M, N 1704 T, N 1704 U) are factory calibrated. Nevertheless a customer calibration, using 3 calibration points, can be processed as described in the following section:

1. Call function „N1700CustomerCalibrateChannelStart“
2. Call „N1700CustomerCalibrateChannelPos“ handing over the channel  
Parameter calPos = -1; calValMM = lower calibration point in unit mm
3. Call „N1700CustomerCalibrateChannelPos“ handing over the channel  
Parameter calPos = 0; calValMM = Zero position
4. Call „N1700CustomerCalibrateChannelPos“ handing over the channel  
Parameter calPos = 1; calValMM = upper calibration point in unit mm
5. Call function „N1700CustomerCalibrateChannelSave“ to save the calibration values inside the module
6. Call function „N1700ActivateCustomerCalibration“ to active the customer calibration



The steps 2, 3 and 4 can be swapped in order, because the values are temporarily stored internally in the N1700.dll. After calling the function „N1700CustomerCalibrateChannelSave“ these temporarily stored values will be used to calculate the calibration parameters „offsetMM“ and „digitsToMM“, which will finally be stored inside the module.

Moreover it is possible to read the calibration parameters „offsetMM“ and „digitsToMM“ and to directly write them independently of above described procedure. Reading the calibration parameter is possible using the function „N1700GetCustomerCalibration“. Writing these parameters can be done using the functions „N1700SetCustomerDigitsToMM“ and „N1700SetCustomerOffsetMM“.

## **10 Examples**

Examples for the following programming languages are available:

C++ (MillimarN1700TestC++)

C# (MillimarN1700TestCSharp und Millimar N 1700LibCSharp)

Delphi (Millimar N 1700 DLL Test Delphi)

## 11 Important Conditions to use the N1700.DLL

1. MAHR Software products are not developed and tested for the high demands in the medical field, in combination with applications in the medical field or in critical components in life-saving systems whose malfunctions or failure can lead to personal injury.
2. On absolutely all applications the stability of the software can be influenced by different factors, i.e. fluctuations in the power supply, computer hardware errors, operating system errors, compiler errors, installation errors, software and hardware compatibility problems, not defined use or misuse or errors by the operator. (All kinds of these errors are called in the following document : SYSTEMERRORS)
3. All applications which contains the risk that SYSTEMERRORS can lead to damages or personal injuries should not only depend on electronic systems. To prevent damages or injuries the operator or system developer should create reasonable precautions against SYSTEMERRORS or their consequents (including backup or shutoff mechanisms).
4. Because all computer systems are adapted for the operator the systems are different in compare to the MAHR test systems. Because the MAHR products can also be integrated in applications not tested or not intended in this way by MAHR the operator or system developer is completely responsible for the test and release of the applications in which MAHR products are embedded. This contains the structure, the procedure and the security level of the application.
5. In no event MAHR will be liable for any damages including lost profits for any special, indirect, incidental or consequential damages arising out of the use or inability to use the product, whether claimed under the safety instructions or otherwise.
6. Corporate guidelines and safety regulations enforced by the industrial trade associations for the prevention of industrial accidents must be strictly observed. Make sure to consult the safety officer at your company.
7. All rights depend on German law.
8. All rights for the N1700.DLL belong to MAHR GmbH

We reserve the right to change the design and technical data contained in our documentation without notifying our customers. MAHR is not obliged to notify changes of the products to previous buyers. All parts of this document must not be reproduced without written permission from MAHR