



# **BA\_3722964-Millimar N1700 DLL Dokumentation\_DE.docx**

---

**Software N1700.DLL**

**Software-Version ab V1.02-11**

13.04.2022

**Bedienungsanleitung**

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 Inhalt

2	Allgemeines .....	4
3	Eigenschaften .....	5
4	Unterstützte Module.....	6
5	Installation unter Windows .....	7
6	Dateien .....	8
7	Überblick über die N1700.DLL .....	9
	Konstanten.....	9
	Konstanten für Rückgabewerte .....	9
	Konstanten für Meldungen.....	9
	Aufzählungstypen: Modul-Typ .....	10
	Aufzählungstypen: Schnittstellen-Typ .....	10
	Aufzählungstypen: Led-Statistiken.....	10
	Aufzählungstypen: Messwert-Datentyp .....	11
	Datenstruktur DLL-Versionen.....	11
	Datenstruktur Modul .....	11
	Datenstruktur Kanal .....	12
	Datenstruktur Erweiterte Kanal Daten .....	12
	Datenstruktur Konfiguration N 1702 VPP .....	12
	Funktionsprototyp Data-Callback .....	13
	Funktionsprototyp Extended Data-Callback .....	13
	Funktionsprototyp Message-Callback.....	13
	Übersicht aller N 1700-Funktionen.....	13
8	Beschreibung der N 1700-Funktionen .....	16
	N1700InitializeLibrary .....	16
	N1700InitializeLibraryNoAutoSearch.....	16
	N1700GetDevices .....	16
	N1700SetAutoSearch.....	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 Erläuterungen zu Modulen .....	28
Parameter des N 1702 VPP Moduls .....	28
Ablauf Kundenkorrektur N 1702/4 M/T/U/M-HR .....	31
10 Beispiele .....	33
11 Important Conditions to use the N1700.DLL .....	34

## **2 Allgemeines**

Die N1700.DLL bietet universelle Treiberzugriffe auf die Millimar N 1700 Module. Sie unterstützt folgende Aufgaben:

- Erkennen der angeschlossenen Module
- Lesen von Einzel-Messwerten
- Starten/Stoppen einer Dauermessung für alle Module
- Lesen und Setzen der Ports von digitalen I/O-Modulen
- Lesen des Fußschalters
- Kalibrierung der Module
- Konfiguration der Module

### 3 Eigenschaften

- Universelle Treiberzugriffe auf Millimar N 1700 Module unter Windows
- Anschluss von Modulen mit insgesamt bis zu 100 Kanälen (max. 64 Module)
- Unterstützt alle Features der Millimar N1700 Module
- Verschiedene Wege, Messwerte abzurufen
  - o Über Polling-Aufruf
  - o Über Messwert-Aufruf in Message-Callback-Funktion
  - o Über Data-Callback-Funktion
- Lesen des Fußschalters über Message-Callback-Funktion bei Änderung des Zustandes des Fußschalters
- Lesen und Setzen der Ports von digitalen I/O-Modulen (N1704 I/O)
- Festlegung der Konfigurationsparameter von Inkremental-Modulen (N 1702 VPP)
- Beispiele für C++, C# und Delphi

## 4 Unterstützte Module

Nachfolgend eine Auflistung, der von den N1700.dll Versionen unterstützten Module

N1700.dll Version	Unterstützte N1700 Module
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 unter Windows

Für die Funktion unter Windows muss sich die Datei **ftd2xx.DLL** im Verzeichnis befinden. Diese DLL wird für die Kommunikation mit dem FTDI-Chip verwendet. Die DLLs **N1700.DLL** bzw. **N1700\_64.DLL** müssen in das Ausführungsverzeichnis der Anwendung kopiert werden.



## 6 Dateien

Die DLL existiert in der Version **N1700.DLL** für 32-Bit-Windows-Anwendungen und in der Version **N1700\_64.DLL** für 64-Bit-Windows-Anwendungen.

Für die Nutzung der DLL in C++ wird die Header-Datei **N1700.H** eingebunden. In Delphi wird die Include-Datei **N1700.INC** eingebunden.

Für C# wird die Bibliothek **MillimarN1700LibCSharp.dll** benötigt, die auch im Quelltext im Beispiel-Projekt **MillimarN1700LibCSharp** vorliegt.

Benötigt wird die Datei **ftd2xx.dll**. Diese ist in der entsprechenden 32-Bit oder 64-Bit Version auch im jeweiligen Ausführungsverzeichnis der Beispielprogramme abgelegt.

## 7 Überblick über die N1700.DLL

### Konstanten

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

### Konstanten für Rückgabewerte

```
#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
```

### Konstanten für Meldungen

```
#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
```

## Aufzählungstypen: 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
};
```

## Aufzählungstypen: Schnittstellen-Typ

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

## Aufzählungstypen: Led-Status

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

## Aufzählungstypen: Messwert-Datentyp

```
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
};
```

## Datenstruktur DLL-Versionen

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

## Datenstruktur Modul

```
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
};
```

Pointer auf Variablen dieser Datenstruktur werden der Funktion N1700GetModule übergeben.

## Datenstruktur Kanal

```

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

```

Pointer auf Variablen dieser Datenstruktur werden der Funktion N1700GetModule übergeben.

## Datenstruktur Erweiterte Kanal Daten

```

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
};

```

## Datenstruktur Konfiguration 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 Position + 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
};

```

## Funktionsprototyp 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 ...
```

## Funktionsprototyp 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);
```

## Funktionsprototyp Message-Callback

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

## Übersicht aller N 1700-Funktionen

Nachfolgend ist eine Übersicht aller zur Verfügung stehenden Funktion dargestellt. Die einzelnen Funktionen werden im Anschluss an dieses Kapitel separat erläutert.

### Allgemeine Funktionen

```
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);
```

### Allgemeine Funktionen für Module und Modul-Parameter

```
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);
```

### Funktionen für Messwertabfrage

```
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);
```

### Funktionen für Werkskalibrierung (nicht zu verwenden!)

```
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);
```

### Funktionen für N 1702 I/O Modul

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

### Funktionen für N 1702 PM/PF Module

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

### Funktionen für N 1702 VPP Modul

```
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 Funktionen

```
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);
```

### **Funktionen für Kunden-Kalibrierung**

```
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 Beschreibung der N 1700-Funktionen

### 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 use the Function **N1700StartContinuousRequestAllData** instead.

```
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.

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

## 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.

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

## 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

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

## 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.

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

## 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 Erläuterungen zu Modulen

### Parameter des N 1702 VPP Moduls

Die Kanäle des N 1702 VPP Moduls werden über die Datenstruktur **sN1700\_CnfVPP** sowie die DLL-Funktionen **F\_N1700GetCnfVPP** und **F\_N1700SetCnfVPP** konfiguriert.

```
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];
};
```

Das N 1702 VPP Modul muss vor der Verwendung konfiguriert werden. Hierfür ist es notwendig, die Parameter der Struktur zu initialisieren. Dies kann entweder direkt über eine Zuweisung der Werte der Struktur erfolgen, oder alternativ über ein Konfigurationsmenü. Die nachfolgende Abbildung zeigt ein Beispiel für ein Konfigurationsmenü (Hier Konfigurationsmenü der Einstellsoftware MillimarN1700.exe). Dieses Menü ermöglicht die Einstellung der Parameter des N 1702 VPP Moduls.

Das dargestellte Menü kann als Anhaltspunkt für eigene Implementierungen verwendet werden. Die Parameter der Struktur `sN1700_CnfVPP` müssen korrekt zugewiesen werden. Die Konfigurationsparameter werden am Beispiel des oben dargestellten Konfigurationsmenüs nachfolgend erläutert.

### Linear oder Rotary Encoder

Von dieser Einstellung hängt der berechnete Messwert ab. Es muss angegeben werden, ob der Sensor linear oder rotativ arbeitet. Je nach Einstellung wird der Messwert in Grad oder Millimeter ausgegeben.

Variable in `sN1700_CnfVPP`: **bool RotaryNotLinear**

0 = Linear Encoder (Beispielsweise Maßstab oder Taster; Messwert in Millimeter)

1 = Rotary Encoder (Drehgeber; Messwert in Grad)

### Periodenlänge oder Inkremente pro Umdrehung

Parameter ist abhängig von der Variable **RotaryNotLinear**. Bei **RotaryNotLinear** = 0 muss die elektrische Periodenlänge in der Einheit Mikrometer angegeben werden. Bei **RotaryNotLinear** = 1 muss die Anzahl der Inkremente pro Umdrehung des angeschlossenen rotativen Messsystems übergeben werden.

Variable in `sN1700_CnfVPP`: **UI32 PerLenOrIncPR**

Wertebereich: 0 bis 65535

### Abstand der Referenzpunkte oder Referenzpunkte pro Umdrehung

Parameter ist abhängig von der Variable **RotaryNotLinear**. Parameter wird nur benötigt, wenn der Multiturnwert aktiviert wird (Parameter **MultiTurn** = 1). Bei **RotaryNotLinear** = 0 muss der Abstand der Referenzpunkte in Mikrometern angegeben werden. Bei **RotaryNotLinear** = 1 muss die Anzahl der Referenzpunkte pro Umdrehung angegeben werden.

Variable in `sN1700_CnfVPP`: **UI32 DistanceRefMarkers**

Wertebereich: 0 bis 65535

### Interpolationsfaktor

Angabe des Interpolationsfaktors. Über diesen Parameter wird die Auflösung über das Verhältnis von elektrischer Periodenlänge zu Interpolationsfaktor bestimmt.

$$\text{Auflösung} = \frac{\text{elektrische Periodenlänge}}{\text{Interpolationsfaktor}}$$

Variable in `sN1700_CnfVPP`: **byte IPolFaktIdx**

Gültige Werte: `IPolFaktIdx` = 0 => Interpolationfaktor = 256

`IPolFaktIdx` = 1 => Interpolationfaktor = 128

`IPolFaktIdx` = 2 => Interpolationfaktor = 64

`IPolFaktIdx` = 3 => Interpolationfaktor = 32

Im oben dargestellten Konfigurationsmenü kann der Interpolationsfaktor beispielsweise über ein Dropdownmenü ausgewählt werden.

## Multiturn oder Singleturn

Es kann ausgewählt werden, ob zusätzlich zum Singleturnwert (Position/Winkel) ebenfalls der Multiturnwert (Anzahl der gezählten Referenzpunkte) berechnet werden soll. Es ist zu beachten, dass bei aktiviertem Multiturnwert intern lediglich mit 22 Bit gerechnet wird. Die gültigen Wertebereiche entnehmen können der separaten Dokumentation des N 1702 VPP Moduls entnommen werden. Die Aktivierung des Multiturnwertes setzt ebenfalls das Setzen der Variable **RefActive** voraus!

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

MultiTurn = 0 => Nur Singleturnwert wird berechnet

MultiTurn = 1 => Singleturnwert + Multiturnwert werden berechnet

## Auswertung Referenzpunkt

Über diesen Parameter kann festgelegt werden, ob der Referenzpunkt des Sensors für die Bestimmung des Messwertes verwendet werden soll. Eine Überfahrt über den Referenzpunkt setzt den Singleturnwert zurück. Ist der Multiturnwert aktiviert, wird bei der Überfahrt ebenfalls der Multiturnwert vorzeichenrichtig inkrementiert.

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

RefActive = 0 => Referenzpunkt wird nicht ausgewertet

RefActive = 1 => Referenzpunkt wird ausgewertet

## Zusätzliche Berechnung der Geschwindigkeit

Zusätzlich zur Berechnung der Position kann ebenfalls die (Winkel-) Geschwindigkeit berechnet werden. Abhängig vom Parameter **RotaryNotLinear** wird entweder eine Geschwindigkeit oder eine Winkelgeschwindigkeit berechnet. Die Zeitbasis für die Berechnung der (Winkel-) Geschwindigkeit beträgt 1 Millisekunde.

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

PosAndVel = 0 => Nur Position/Winkel

PosAndVel = 1 => Position + (Winkel-)Geschwindigkeit

Im oben dargestellten Konfigurationsmenü werden für diese Einstellung Radiobutton mit den Bezeichnungen „Position“ sowie „Position + Velocity“ verwendet.

## Messwertfilterung

Beim N 1702 VPP Modul besteht die Möglichkeit der analogen Filterung der Sin/Cos-Eingangssignale. Über den Parameter **FilterOn** kann diese Filterung aktiviert werden. Mit dem Parameter **TpFreqIdx** wird die Grenzfrequenz dieses Tiefpassfilters festgelegt. Es kann zwischen den Frequenzen 450kHz, 200kHz, 75kHz und 10kHz ausgewählt werden.

Variablen 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

## Korrektur der Phasenverschiebung von Sin und Cos

**Die nachfolgenden Parameter dienen internen Testzwecken und dürfen nicht zugewiesen werden! Die Funktionen hinter den Variablen sind bisher nicht freigegeben und somit nicht nutzbar!**

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

PhasenCorrOn = 0 => Korrektur der Phasenverschiebung deaktiviert  
PhasenCorrOn = 1 => Korrektur der Phasenverschiebung aktiviert

PhasenCorrOk = 0 => Korrektur wurde noch nie ausgeführt  
PhasenCorrOk = 1 => Korrektur wurde in Vergangenheit ausgeführt

PhaseCorrDeg10Value => Wert der Phasenverschiebung (Wertebereich -4,9° bis 4,9°)

## Parameter RefStatOk

Der Parameter **RefStatOk** gibt an, ob eine Referenzierung am Referenzpunkt bereits stattgefunden hat. Die Referenzierung findet bei der ersten Überfahrt eines Referenzpunktes statt. Der Parameter **RefStatOk** wird bei jedem Abruf eines Messwertes des N 1702 VPP Moduls aktualisiert. Der Parameter **RefStatOk** wird nur aktualisiert, wenn der Parameter **RefActive** in der Struktur gesetzt ist.

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

RefStatOk = 0 => Referenzierung (noch) nicht erfolgt  
RefStatOk = 1 => Am Referenzpunkt wurde referenziert

## Ablauf Kundenkorrektur N 1702/4 M/T/U/M-HR

Module zum Anschluss von induktiven Messtastern (N 1702 M, N 1702 T, N 1702 U, N 1702 M-HR, N 1704 M, N 1704 T, N 1704 U) sind ab Werk kalibriert. Dennoch besteht die Möglichkeit einer eignen Kalibrierung, der Kundenkalibrierung. Eine 3-Punkt Kundenkalibrierung kann wie nachfolgend beschrieben durchgeführt werden:

1. Aufruf der Funktion „N1700CustomerCalibrateChannelStart“
2. „N1700CustomerCalibrateChannelPos“ mit Angabe des zu kalibrierenden Kanals  
Parameter calPos = -1; calValMM = unterer Kalibrierpunkt in Einheit Millimeter
3. Aufruf der Funktion „N1700CustomerCalibrateChannelPos“  
Parameter calPos = 0; calValMM = Nullpunkt
4. „N1700CustomerCalibrateChannelPos“ mit Angabe des zu kalibrierenden Kanals  
Parameter calPos = 1; calValMM = oberer Kalibrierpunkt



5. Aufruf der Funktion „N1700CustomerCalibrateChannelSave“ zum Speichern der Kalibrierwerte auf dem Modul
6. Aufruf der Funktion „N1700ActivateCostumerCalibration“ zur Aktivierung der Kundenkorrektur

Die Schritte 2, 3 und 4 können auch in anderer Reihenfolge durchgeführt werden, da diese Werte intern in der DLL zwischengespeichert werden. Erst bei Aufruf der Funktion „N1700CustomerCalibrateChannelSave“ werden aus den Kalibrierpunkten die Kalibrier-Parameter „offsetMM“ und „digitsToMM“ berechnet und gespeichert.

Weiterhin besteht die Möglichkeit die Kalibrier-Parameter „offsetMM“ und „digitsToMM“ auszulesen und diese separat zu schreiben. Zum Auslesen der Parameter der Kundenkalibrierung kann die Funktion „N1700GetCustomerCalibration“ verwendet werden. Das separate Schreiben der Parameter ist mit den Funktionen „N1700SetCustomerDigitsToMM“ und „N1700SetCustomerOffsetMM“ möglich.

## **10 Beispiele**

Es existieren Beispiele für folgende Programmier-Sprachen:

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