

Irinos EC

© 2019-2020 Messtechnik Sachs GmbH

Diese Betriebsanleitung wurde für die Darstellung in einem Webbrowser im HTML-Format optimiert. Verwenden Sie die PDF-Version nur, wenn kein Zugriff auf die Online-Hilfe möglich ist.

1. Übersicht	9
2. Irinos EC Benutzerhandbuch	11
2.1 Einleitung	12
2.1.1 Revisions-Historie	13
2.1.2 Rechtliche Hinweise	13
2.1.2.1 Nutzungshinweise für Software / elektronische Dokumentation	13
2.1.2.2 Warnhinweiskonzept	17
2.1.2.3 Qualifiziertes Personal	18
2.1.2.4 Haftungsausschluss	18
2.1.3 Vorwort	19
2.2 Sicherheitshinweise	21
2.3 System-Übersicht	26
2.3.1 Modularität	27
2.3.2 Synchronisation und Geschwindigkeit	29
2.3.3 Master vs. Slave	30
2.3.4 Spannungsversorgung	31
2.4 Produktbeschreibungen	31
2.4.1 Grundaufbau Irinos-EC - Box mit EC-Link - Schnittstelle	34
2.4.2 EC-TFV für induktive Wegaufnehmer / Messtaster	37
2.5 Steckerbelegungen	39
2.5.1 Spannungsversorgung 24V	39
2.5.2 Ethernet	41
2.6 Montage	41
2.6.1 Lieferumfang / Prüfen der Lieferung	42
2.6.2 Auswahl des Standorts	43
2.6.3 Befestigung	44
2.6.4 Leitungen anschließen	46
2.6.4.1 EC-Link Verkabelung	47
2.6.4.2 Ethernet anschließen	48
2.6.4.3 Spannungsversorgung anschließen	49
2.6.5 Messmodule einbauen	51
2.7 Inbetriebnahme und Kennenlernen	51
2.7.1 Box-Adressierung	52
2.7.2 Netzwerk-Konfiguration	53
2.7.3 Irinos-Tool	54
2.7.4 Web-Server	55
2.8 Software-Schnittstelle	60
2.8.1 NmxDLL Kurzübersicht	62
2.8.2 ASCII- / Telnet-Schnittstelle	63
2.8.3 MscDLL Kurzübersicht	67
2.9 Diagnose und "Erste Hilfe"	67
2.9.1 Diagnose-Ereignisse	67

2.9.2	Diagnose-Speicher	77
2.9.3	Erste Hilfe "Netzwerkverbindung"	78
2.10	Wartung, Pflege und Entsorgung	80
2.11	Applikationshinweise	82
2.11.1	Inkrementalgeber	82
2.11.1.1	Referenzierung bei Absolutmessung	82
2.11.1.2	Eingangsfrequenz	83
2.11.1.3	Interpolation (nur 1Vss)	83
2.11.2	Leistungsaufnahme	86
2.11.3	Speichern im nicht-flüchtigen Speicher	86
2.12	Technische Daten	87
2.12.1	Allgemeine technische Daten	88
3.	Irinos-Tool Benutzerhandbuch	91
3.1	Einleitung	92
3.1.1	Impressum	92
3.1.2	Revisions-Historie	92
3.1.3	Glossar	94
3.1.4	Nutzungshinweise für Software / elektronische Dokumentation	94
3.1.5	Vorwort	98
3.1.5.1	Zweck	98
3.1.5.2	Gültigkeitsbereich dieser Betriebsanleitung	98
3.1.5.3	Bestimmungsgemäßer Gebrauch	98
3.1.5.4	Erforderliche Grundkenntnisse	99
3.1.5.5	Weitere Dokumentation	99
3.1.5.6	Versionsstand	99
3.2	Einführung	99
3.2.1	Über diese Hilfe	99
3.2.2	Übersicht	100
3.3	Kurzanleitung	101
3.3.1	Voraussetzungen	101
3.3.2	Einstellungen an der PC-Netzwerkkarte	102
3.3.3	Irinos-Konfiguration und Verbindungsprüfung	103
3.4	PC-seitige Netzwerkanbindung	106
3.4.1	Ethernet-Verbindung	106
3.4.2	Netzwerkschnittstellen	107
3.4.3	Netzwerkeinstellungen	110
3.4.3.1	IP-Konfiguration mit DHCP	110
3.4.3.2	IP-Konfiguration ohne DHCP	112
3.5	Irinos-Tool	114
3.5.1	Allgemeines	114
3.5.2	Installation	114
3.5.3	Starten des Irinos-Tools	114
3.5.4	IP-Konfiguration	115
3.5.5	Direkte IP-Eingabe	118

3.5.6	Verbindungsprüfung über DLL	119
3.5.7	Kanalzuordnung	121
3.5.7.1	Auswahl der Eingangstyps bei Inkrementalgebern	121
3.5.7.2	Kanalzuordnung ändern	122
3.5.8	Inventardaten	123
3.5.8.1	Absolutzeit setzen	125
3.5.8.2	Ereigniskonfiguration	126
3.5.9	Statische Messung	127
3.5.10	Dynamische Messung	128
3.5.11	Digitale Ein-/Ausgänge	129
3.5.12	Diagnose-Speicher	130
3.5.13	Firmware-Update	131
3.5.13.1	Versionsnummern	131
3.5.13.2	Update ausführen	132
3.5.14	Inkrementalgeber-Diagnose	135
3.5.14.1	Live-Anzeige (nur 1Vss)	135
3.5.14.2	Historie (nur 1Vss)	142

4. NmxDLL Referenz 149

4.1	Introduction	150
4.1.1	Imprint	150
4.1.2	Revision history	150
4.1.3	Legal notes	151
4.1.3.1	Terms of use for documentation & software	151
4.1.3.2	Qualified personnel	154
4.1.3.3	Disclaimer	154
4.1.4	Preface	154
4.1.4.1	Purpose	154
4.1.4.2	Scope of this reference manual	154
4.1.4.3	Required knowledge	154
4.1.4.4	Further documentation	154
4.2	Nmx DLL Overview	155
4.2.1	Static vs. Sampling	156
4.2.2	Sampling Speed with Irinos	157
4.2.3	Data Types	164
4.2.4	Technical Background	165
4.2.5	Limitations	165
4.2.6	Hardware Requirements	166
4.2.7	Versions	166
4.2.8	INI-File	168
4.2.9	.NET Wrapper DLL	168
4.3	API (programming interface)	169
4.3.1	Function calls overview	169
4.3.2	Function Return Codes (NMX_STATUS)	176
4.3.3	Connection Handle	178
4.3.4	Trigger Modes	179

4.3.5	Miscellaneous	180
4.3.5.1	NMX_GetDllVersion_1	180
4.3.5.2	NMX_SystemReset_1	181
4.3.5.3	NMX_ChannelSetParameter_1	182
4.3.5.4	NMX_ChannelSetConfig_1	186
4.3.6	Connecting / Disconnecting	189
4.3.6.1	NMX_DeviceIPv4Open_1	189
4.3.6.2	NMX_DeviceClose_1	191
4.3.7	Notifications	192
4.3.7.1	NMX_RegisterMessage_1	194
4.3.7.2	NMX_RegisterCallback_1	196
4.3.8	Get device information	199
4.3.8.1	NMX_GetBoxCount_1	199
4.3.8.2	NMX_GetBoxInfo_1	200
4.3.8.3	NMX_UpdateChannelInfo_1	204
4.3.8.4	NMX_GetChannelCount_1	205
4.3.8.5	NMX_GetChannelInfo_1	206
4.3.8.6	NMX_GetDigitalInputInfo_1	211
4.3.8.7	NMX_GetDigitalOutputInfo_1	213
4.3.9	Static Measurement (Non-Realtime)	214
4.3.9.1	NMX_StaticGet32_1	214
4.3.9.2	NMX_StaticSetMedianDepth_1	220
4.3.9.3	NMX_SetOutputs_1	221
4.3.9.4	NMX_DisableOutputUpdate_1	222
4.3.9.5	NMX_DigitalIoConfig_1	223
4.3.9.6	NMX_DigitalOutputsGetState_1	224
4.3.10	Sampling LowLevel (Time-Triggered Realtime Measurement)	225
4.3.10.1	NMX_Sampling_GetMaxSpeed_1	225
4.3.10.2	NMX_Sampling_Reset_1	226
4.3.10.3	NMX_Sampling_AddChannelsAll_1	227
4.3.10.4	NMX_Sampling_AddChannel_1	228
4.3.10.5	NMX_Sampling_AddDigiInAll_1	230
4.3.10.6	NMX_Sampling_AddDigiInByte_1	231
4.3.10.7	NMX_Sampling_AddDigiOutAll_1	232
4.3.10.8	NMX_Sampling_AddDigiOutByte_1	233
4.3.10.9	NMX_Sampling_PrepareTime_1	234
4.3.10.10	NMX_Sampling_Start_1	236
4.3.10.11	NMX_Sampling_Stop_1	237
4.3.10.12	NMX_Sampling_ReadColumn32_1	237
4.3.10.13	NMX_Sampling_ReadRow32_1	240
4.3.10.14	NMX_Sampling_GetStatus_1	242
4.3.11	Sampling HighLevel (Application-specific Realtime Measurement)	244
4.3.11.1	NMX_Sampling_PreparePosition_1	245
4.3.11.2	NMX_Sampling_PrepareCustomTFT_1	248
4.3.12	Diagnostics	251
4.3.12.1	NMX_DiagClearEvent_1	251
4.3.12.2	NMX_DiagGetEventText_1	252
4.3.12.3	NMX_SetDateTime_1	254
4.4	HowTo	255
4.4.1	Small Measurement Application	255

- 4.4.2 Establishing a connection 255
- 4.4.3 Closing a connection 257
- 4.4.4 Reading static data 258
 - 4.4.4.1 Cyclically (Polling) 259
 - 4.4.4.2 Event based 261
- 4.4.5 Sampling 266
 - 4.4.5.1 Start endless time-based sampling 268
 - 4.4.5.2 Start time-limited sampling 275
 - 4.4.5.3 Stop sampling 282
 - 4.4.5.4 Reading sampled data 283
 - 4.4.5.4.1 Read Column-Wise 284
 - 4.4.5.4.2 Read Row-Wise 289
 - 4.4.5.5 Get sampling status 293
 - 4.4.5.5.1 Poll sampling status 293
 - 4.4.5.5.2 Sampling notifications 294
 - 4.4.5.6 Start position triggered sampling 300
 - 4.4.5.7 Start TFT high-level sampling 304

Index

309

Übersicht

1 Übersicht

Diese Web-basierte Hilfe besteht aus 3 Handbüchern:

A. [Irinos EC Benutzerhandbuch](#)  26

Dies ist die **Haupt-Dokumentation**. Beginnen Sie hier.

B. [Irinos Tool Bedienungsanleitung](#)  101

In dieser Dokumentation ist die Bedienung der Inbetriebnahme und Diagnose-Software Irinos-Tool bzw. ITool beschrieben.

C. [NmxDLL Referenz \(Software API\)](#)  155

Beschreibung der Software-Schnittstelle der NMX DLL. Für Software-Entwickler.

Irinos EC Benutzerhandbuch

2 Irinos EC Benutzerhandbuch

2.1 Einleitung

Diese Bedienungsanleitung erläutert die Inbetriebnahme, Montage und Handhabung des Irinos EC - Messsystems. Lesen Sie diese vor Verwendung des Produkts.

Sie wurde für die Betrachtung auf elektronischen Endgeräten optimiert und enthält multimediale Inhalte. Es wird deshalb empfohlen die elektronische Version mit einem PC, Tablet, Smartphone oder Ähnlichem zu verwenden.

- Zu Beginn dieser Bedienungsanleitung finden Sie diese Einleitung samt zugehörigen [rechtlichen Hinweisen](#)^[13], sowie die [Sicherheitshinweise](#)^[21].
- Die [System-Übersicht](#)^[26] gibt eine generelle Einführung in das Irinos EC - System. Diese wird ergänzt durch [Produktbeschreibungen](#)^[31] sowie die zugehörigen [Stecker-Belegungen](#)^[39].
- Darauf folgen Informationen zur [Montage](#)^[41] und zur [Inbetriebnahme](#)^[51] des Systems.
- Ergänzend finden Sie Informationen zur [Diagnose](#)^[67], zu [Wartung, Pflege und Entsorgung](#)^[80] sowie spezifische [Applikationshinweise](#)^[82].

Impressum

Titel	Irinos EC
Hersteller	Messtechnik Sachs GmbH Siechenfeldstraße 30/1 D-73614 Schorndorf Tel. 07181 / 26935-0 post@messtechnik-sachs.de
Gültig für	Messmodule Irinos EC
Copyright-Hinweis	© 2019-2020 Messtechnik Sachs GmbH
Hinweis auf Markenzeichen und Warenzeichen	Alle in diesem Handbuch genannten Bezeichnungen von Erzeugnissen sind Warenzeichen der jeweiligen Firmen.
Material-Nr.	785-1028
Änderungshinweis	Technische Änderungen vorbehalten.
Stand	05.06.2020

2.1.1 Revisions-Historie

Version	Datum	Änderungen
A	2019-08-19	Erste Version

2.1.2 Rechtliche Hinweise

2.1.2.1 Nutzungshinweise für Software / elektronische Dokumentation

Schutzrechte und Nutzungsumfang

Messtechnik Sachs stellt entweder auf portablen Datenträgern (z. B. Disketten, CD-ROMs, DVDs, ...), in schriftlicher (drucktechnischer) oder elektronischer Form Bedienungsanleitungen, Handbücher, Dokumentationen, sowie

Softwareprogramme, alles und insgesamt im Folgenden als "LIZENZGEGENSTAND" bezeichnet, entgeltlich und/oder unentgeltlich zur Verfügung. Der LIZENZGEGENSTAND unterliegt u.a. urheberrechtlichen Schutzbestimmungen. Messtechnik Sachs oder Dritte haben Schutzrechte an diesem LIZENZGEGENSTAND. Soweit Dritten ganz oder teilweise Rechte an diesem LIZENZGEGENSTAND zustehen, hat Messtechnik Sachs entsprechende Nutzungsrechte. Messtechnik Sachs gestattet dem Verwender die Nutzung des LIZENZGEGENSTANDES unter den folgenden Voraussetzungen:

1.1) Nutzungsumfang elektronische Dokumentation

- a) Mit dem Erhalt/Erwerb oder der Überlassung eines LIZENZGEGENSTANDES erhalten Sie als Verwender in Bezug auf den jeweiligen LIZENZGEGENSTAND ein einfaches, nicht übertragbares Nutzungsrecht, das den Verwender berechtigt, diesen für eigene, ausschließlich betriebsinterne Zwecke, auf beliebig vielen Maschinen innerhalb seines Betriebsgeländes (Einsatzort), zu nutzen. Dieses Nutzungsrecht umfasst ausschließlich das Recht, den LIZENZGEGENSTAND auf den am Einsatzort eingesetzten Zentraleinheiten (Maschinen) zu speichern.
- b) Bedienungsanleitungen und/oder Dokumentationen, ungeachtet in welcher Form zur Verfügung gestellt, darf der Verwender an dessen Einsatzort außerdem in beliebiger Zahl über einen Drucker ausdrucken, sofern dieser Ausdruck vollständig mit diesen Nutzungsbedingungen und sonstigen Benutzerhinweisen ausgedruckt bzw. verwahrt wird.
- c) Mit Ausnahme des Messtechnik Sachs Logos ist der Verwender berechtigt, Bilder und Texte der Bedienungsanleitungen/Dokumentationen zur Erstellung eigener Maschinen- und Anlagendokumentation zu verwenden. Die Verwendung des Messtechnik Sachs Logos bedarf der schriftlichen Genehmigung von Messtechnik Sachs. Für die Übereinstimmung genutzter Bilder und Texte mit der Maschine/Anlage bzw. dem Produkt ist der Verwender selbst verantwortlich.
- d) Weitergehende Nutzungen sind in folgendem Rahmen zulässig:

Das Vervielfältigen ausschließlich zur Verwendung im Rahmen einer Maschinen- und Anlagendokumentation aus elektronischen Dokumenten sämtlicher dokumentierter Zulieferbestandteile. Die Demonstration gegenüber Dritten ausschließlich unter Sicherstellung, dass kein Datenmaterial ganz oder teilweise in anderen Netzwerken oder anderen Datenträgern verbleibt oder dort reproduziert werden kann.

Die Weitergabe von Ausdrucken an Dritte außerhalb der Regelung in Ziffer 3 sowie jede Bearbeitung oder andersartige Verwendung sind nicht zulässig.

1.2) Nutzungsumfang Softwareprodukte

An Software von Messtechnik Sachs jeglicher Art und der dazugehörigen Dokumentation erhält der Kunde ein nicht ausschließliches, nicht übertragbares und zeitlich nicht begrenztes Nutzungsrecht auf einem bestimmten bzw. im Einzelfall festzulegenden Hardware-Produkt. Messtechnik Sachs bleibt Inhaberin des Urheberrechts sowie aller anderen gewerblichen Schutzrechte. Das Recht Vervielfältigungen anzufertigen, ist nur zum Zwecke der Datensicherung gegeben. Copyright-Vermerke dürfen nicht entfernt werden.

2. Copyright Vermerk

Jeder LIZENZGEGENSTAND enthält einen Copyright Vermerk. Bei jeglicher Vervielfältigung die nach diesen Bestimmungen erlaubt ist, muss der entsprechende Copyright Vermerk des betreffenden Originals übernommen werden:

Bsp.: © 2015-2019, Messtechnik Sachs GmbH,
D-73614 Schorndorf

3. Übertragung der Nutzungsbefugnis

Der Verwender kann seine Nutzungsbefugnis nach diesen Bestimmungen bzgl. des jeweiligen LIZENZGEGENSTANDES in dem Umfang und mit den Beschränkungen der Bedingungen gemäß Ziffer 1 und 2 insgesamt auf einen Dritten übertragen. Auf diese Nutzungsbedingungen ist der Dritte ausdrücklich hinzuweisen.

II. Export LIZENZGEGENSTAND

Der Verwender muss beim Export des LIZENZGEGENSTANDES oder Teilen davon die Ausfuhrbestimmungen des ausführenden Landes und des Landes des Erwerbs beachten.

III. Gewährleistung

1. Produkte von Messtechnik Sachs werden hard- und softwaretechnisch weiterentwickelt. Liegt der LIZENZGEGENSTAND, gleich in welcher Form, einem Produkt nicht unmittelbar bei, d.h. wird nicht auf einem, dem Produkt beiliegenden portablen Datenträger mit dem betreffenden Produkt als Liefereinheit ausgeliefert, gewährleistet Messtechnik Sachs nicht, dass eine elektronische Dokumentation mit jedem Hard- und Software-Stand des Produkts übereinstimmt.

2. Die in einer elektronischen Dokumentation enthaltenen Informationen können von Messtechnik Sachs ohne Vorankündigungen geändert werden und stellen keine Verpflichtung seitens Messtechnik Sachs dar.

3. Messtechnik Sachs gewährleistet, dass das von ihr erstellte Softwareprogramm mit der Anwendungsbeschreibung und Programmspezifikation übereinstimmt, jedoch nicht, dass die in der Software enthaltenen Funktionen vollständig unterbrechungs- u. fehlerfrei laufen oder dass die in der Software enthaltenen Funktionen in allen vom Erwerber gewählten Kombinationen und vorgesehenen Einsatzbedingungen ausführbar sind, bzw. den Erfordernissen entsprechen.

IV. Haftung/Haftungsbeschränkungen

1. Messtechnik Sachs stellt LIZENZGEGENSTÄNDE zur Verfügung, um den Verwender einerseits in die Lage zu versetzen Messtechnik Sachs Produkte die zum ordnungsgemäßen Betrieb einer Software bedürfen, diese vertragsgemäß einzusetzen, oder ihn bei der Erstellung seiner Maschinen- und Anlagendokumentation zu unterstützen. Für die elektronische Dokumentation, die in Form von portablen Datenträgern nicht unmittelbar einem Produkt beiliegt, d.h. nicht mit einem Produkt als Liefereinheit ausgeliefert wurde, gewährleistet

Messtechnik Sachs garantiert jedoch nicht, dass die separat vorgehaltene/gelieferte elektronische Dokumentation mit dem vom Verwender tatsächlich genutzten Produkt übereinstimmt.

Letzteres gilt insbesondere bei auszugsweisem Gebrauch für eigene Dokumentationen des Verwenders. Die Gewährleistung und Haftung für separat vorgehaltene / gelieferte portable Datenträger, d.h. mit Ausnahme der im Internet/Intranet vorgehaltenen elektronischen Dokumentation, beschränkt sich ausschließlich auf eine ordnungsgemäße Duplikation der Software, wobei Messtechnik Sachs gewährleistet, dass jeweils der neueste Stand der Dokumentation Inhalt des betreffenden, portablen Datenträgers ist. In Bezug auf die im Internet/Intranet vorgehaltene elektronische Dokumentation wird nicht gewährleistet, dass diese denselben Versionsstand aufweist wie die zuletzt drucktechnisch veröffentlichte Ausgabe.

2. Messtechnik Sachs haftet ferner nicht für mangelnden wirtschaftlichen Erfolg oder für Schäden oder Ansprüche Dritter wegen der Nutzung/Verwendung der vom Verwender eingesetzten LIZENZGEGENSTÄNDE, mit Ausnahme von Ansprüchen aus der Verletzung von Schutzrechten Dritter, welche die Nutzung der LIZENZGEGENSTÄNDE betreffen.

3. Die Haftungsbeschränkungen nach Absatz 1. und 2. gelten nicht, soweit in Fällen von Vorsatz oder grober Fahrlässigkeit oder Fehlen zugesicherter Eigenschaften eine zwingende Haftung

besteht. In einem solchen Fall ist die Haftung von Messtechnik Sachs auf den Schaden begrenzt, der für Messtechnik Sachs nach der Kenntnis der konkreten Umstände erkennbar war.

V. Sicherheitsrichtlinien/Dokumentation

Gewährleistungs- und Haftungsanspruch nach Maßgabe der vorstehenden Regelungen (Ziff. III. u. IV) sind nur gegeben, wenn der Anwender die Sicherheitsrichtlinien einer Dokumentation im Zusammenhang mit der Nutzung der Maschine und deren Sicherheitsrichtlinien oder die Nutzungsbedingungen von Software beachtet hat. Für die Kompatibilität nicht mit einem Produkt als Liefereinheit ausgelieferter elektronischer Dokumentation mit dem vom Anwender tatsächlich genutzten Produkt ist der Anwender selbst verantwortlich.

2.1.2.2 Warnhinweiskonzept

Diese Bedienungsanleitung enthält Hinweise, die Sie zu Ihrer persönlichen Sicherheit sowie zur Vermeidung von Sachschäden beachten müssen. Die Hinweise zu Ihrer persönlichen Sicherheit sind durch ein Warndreieck hervorgehoben, Hinweise zu alleinigen Sachschäden stehen ohne Warndreieck. Je nach Gefährdungstufe werden die Warnhinweise in abnehmender Reihenfolge wie folgt dargestellt.

	Gefahr
	bezeichnet eine unmittelbar drohende Gefahr. Wenn sie nicht gemieden wird, sind Tod oder schwerste Verletzungen (Verkrüppelung) die Folge.

	Warnung
	bezeichnet eine möglicherweise gefährliche Situation. Wenn sie nicht gemieden wird, können Tod oder schwerste Verletzungen die Folge sein.

	Vorsicht
	bezeichnet eine möglicherweise gefährliche Situation. Wenn sie nicht gemieden wird, können leichte oder geringfügige Verletzungen die Folge sein.

Achtung
bezeichnet eine möglicherweise schädliche Situation. Wenn sie nicht gemieden wird, kann das Produkt oder etwas in der Umgebung beschädigt werden.

2.1.2.3 Qualifiziertes Personal

Das zu dieser Dokumentation zugehörige Produkt/System darf nur von für die jeweilige Aufgabenstellung qualifiziertem Personal gehandhabt werden unter Beachtung der für die jeweilige Aufgabenstellung zugehörigen Dokumentation, insbesondere der darin enthaltenen Sicherheits- und Warnhinweise. Qualifiziertes Personal ist auf Grund seiner Ausbildung und Erfahrung befähigt, im Umgang mit diesen Produkten/Systemen Risiken zu erkennen und mögliche Gefährdungen zu vermeiden.

2.1.2.4 Haftungsausschluss

Der Inhalt dieser Dokumentation wurde von uns sorgfältig auf Übereinstimmung mit der beschriebenen Hard- und Software geprüft. Abweichungen können dennoch nicht ausgeschlossen werden, so dass wir für die vollständige Übereinstimmung keine Gewähr übernehmen. Die Angaben in dieser Dokumentation werden regelmäßig geprüft. Etwaige Korrekturen sind in den nachfolgenden Auflagen enthalten.

2.1.3 Vorwort

	Warnung
	<p>Lesen Sie dieses Benutzerhandbuch sowie die zugehörige Dokumentation vor der Inbetriebnahme und Nutzung des Irinos-Systems vollständig durch. Dies gilt insbesondere für die darin enthaltenen Sicherheitshinweise.</p> <p>Fehlanwendung kann zu Schaden an Mensch, Maschine oder Anlage führen.</p>

Zweck

Diese Bedienungsanleitung enthält alle erforderlichen Informationen für die Inbetriebnahme, die Nutzung sowie die Wartung des Irinos EC - Messsystems. Zur Zielgruppe gehören Benutzer und Servicetechniker, die das Produkt in Betrieb nehmen, es parametrieren oder Fehleranalysen durchführen.

Gültigkeitsbereich dieser Betriebsanleitung

Diese Betriebsanleitung gilt für das industrielle Messsystem Irinos EC sowie den zugehörigen Optionen.

Bestimmungsgemäßer Gebrauch

Irinos EC ist ein flexibles High-Speed - Messsystem für die industrielle Fertigungsmesstechnik. Es ist für den Dauerbetrieb (24/7) geeignet.

Das Messgerät ist ausdrücklich nicht geeignet für den Einsatz in medizinischen oder explosionsgefährdeten Bereichen, Flugzeugen, für die Raumfahrt sowie für den Heim- und Bürobereich. Nicht aufgeführte Bereiche, die diesen vom Sinn her ähnlich sind, gehören ebenfalls dazu.

In sicherheitskritischen Bereichen ist die Betriebssicherheit durch externe Vorrichtungen zu gewährleisten (z.B. externer Not-Aus-Kreis).

Bitte beachten Sie:

	Warnung
	Produkte der Messtechnik Sachs GmbH dürfen nur für die im Datenblatt und in der zugehörigen technischen Dokumentation vorgesehenen Einsatzfälle verwendet werden. Falls Fremdprodukte und –komponenten zum Einsatz kommen, müssen diese von der Messtechnik Sachs GmbH empfohlen bzw. zugelassen sein. Der einwandfreie und sichere Betrieb der Produkte setzt sachgemäßen Transport, sachgemäße Lagerung, Aufstellung, Montage, Installation, Inbetriebnahme, Bedienung und Instandhaltung voraus. Die zulässigen Umgebungsbedingungen müssen eingehalten werden. Hinweise in den zugehörigen Dokumentationen müssen beachtet werden.

Erforderliche Grundkenntnisse

Für die mechanische Integration und Befestigung sind solide Kenntnisse in den Bereichen Mechanik und Maschinenbau erforderlich.

Für die elektrische Installation und die Inbetriebnahme sind Fachkenntnisse in Elektrotechnik sowie elektrotechnischer Sicherheit erforderlich.

Für die Einrichtung der Messaufgabe sind fundierte Kenntnisse in der industriellen Messtechnik sowie PC-Kenntnisse erforderlich.

Weitere Dokumentationen

Beachten Sie das Begleitblatt, das mit jedem Irinos-Modul mitgeliefert wird. Dies gilt insbesondere für die darin enthaltenen Sicherheitshinweise. Die technischen Daten sind dem jeweils zugehörigen Datenblatt zu entnehmen.

Für das zugehörige Diagnose- und Parametrierwerkzeug "ITool" steht eine separate Dokumentation zur Verfügung.

Entwickler finden die Beschreibung zur Nutzung der Kommunikations-Bibliothek im Referenz-Handbuch für diese Bibliothek.

Versionsstand der Firmware

Diese Bedienungsanleitung bezieht sich auf den Firmware Versionsstand 2.

(Version 2 ist die erste verfügbare Version. Die Versionsnummer 1 wurde zur besseren Übersichtlichkeit mit anderen Systemen übersprungen.)

2.2 Sicherheitshinweise

Achtung

Beschädigung durch Öffnen des Gerätes

Öffnen Sie die Irinos-Komponenten nicht. Sie sind so konzipiert, dass ein Öffnen nicht erforderlich ist. Die Messbox und/oder das Messsystem könnten beschädigt werden. Fehlfunktionen oder Zerstörung wären die mögliche Folge.

Durch Öffnen der Irinos-Komponenten erlischt die Gewährleistung.

Achtung

Ungewollte Betriebssituation

Hochfrequente Strahlung, z.B. vom Mobiltelefon, stört Gerätefunktionen und kann zu Fehlfunktionen des Irinos-Messsystems führen.

Personen können verletzt und die Anlage beschädigt werden.

Vermeiden Sie hochfrequente Strahlung:

- Entfernen Sie Strahlungsquellen aus der Umgebung des Irinos Messsystems.
- Schalten Sie strahlende Geräte ab.
- Reduzieren Sie die Funkleistung strahlender Geräte.

Stellen Sie die Einhaltung der elektromagnetischen Verträglichkeit sicher.

	Warnung
	Elektrischer Schlag
	Eine unzureichende Erdung und/oder Trennung vom Versorgungsnetz kann zu einem Schaden von Mensch, Maschine und Anlage führen.
	Bitte beachten Sie:
	<ul style="list-style-type: none">○ Verwenden Sie für die elektrische Versorgung ausschließlich PELV-Stromkreise nach IEC60204-1 (Protective Extra-Low Voltage, PELV).○ Berücksichtigen Sie zusätzlich die allgemeinen Anforderungen an PELV-Stromkreise nach IEC60204-1.
	Verwenden Sie ausschließlich Spannungsquellen, die eine sichere elektrische Trennung der Betriebs- und Lastspannung nach IEC60204-1 gewährleisten.

	Warnung
	Gefährdungen an der ungeschützten Maschine oder Anlage
	Bei Betrieb in einer Maschine oder Anlage ist zu beachten: An der ungeschützten Maschine können entsprechend der Ergebnisse einer Risikoanalyse Gefährdungen bestehen. Diese Gefährdungen können zu Personenschaden führen.
	Die Gefährdung von Personen vermeiden Sie entsprechend der Risikoanalyse durch folgende Maßnahmen: <ul style="list-style-type: none">○ Zusätzliche Schutzeinrichtungen an der Maschine oder Anlage. Hierbei müssen insbesondere Programmierung, Parametrierung und Verdrahtung der eingesetzten Peripherie entsprechend der durch notwendige Risikoanalyse festgestellten Sicherheitsperformance (SIL, PL oder Kat.) erfolgen.○ Die bestimmungsgemäße Verwendung des Irinos-Messsystems, die Sie durch einen Funktionstest an der Anlage nachweisen. Damit können Programmier-, Parametrier- und Verdrahtungsfehler erkannt werden.○ Dokumentation der Testergebnisse, die Sie bei Bedarf in die relevanten Sicherheitsnachweise eintragen.

	Achtung
	Elektrostatisch gefährdete Bauteile
	Eine Irinos-Box enthält elektrostatisch gefährdete Bauteile. Diese können bereits beim Berühren durch Spannungen zerstört werden, die weit unterhalb der Wahrnehmungsgrenze des Menschen liegen. Öffnen Sie eine Irinos-Box nicht. Sie vermeiden dadurch eine Berührung der gefährdeten Bauteile.

Achtung**Beschädigung des Irinos-Messsystems bei Transport und Lagerung**

Wenn ein Irinos-Messsystem ohne Verpackung transportiert oder gelagert wird, wirken Stöße, Schwingungen, Druck und Feuchtigkeit ungeschützt auf die Irinos-Komponenten ein. Eine beschädigte Verpackung weist darauf hin, dass Umgebungsbedingungen bereits massiv auf die Irinos-Komponenten eingewirkt haben.

Die Irinos-Komponenten und/oder das Irinos-System kann/können beschädigt werden.

Entsorgen Sie nicht die Originalverpackung. Verpacken Sie die Irinos-Komponenten bei Transport und Lagerung.

Achtung**Beschädigung durch Betauung**

Wenn die Irinos-Komponenten bzw. das Irinos-Messsystem während des Transports niedrigen Temperaturen oder extremen Temperaturschwankungen ausgesetzt wurde, z.B. bei kalter Witterung, kann sich Feuchtigkeit am oder in den Irinos-Komponenten niederschlagen (Betauung).

Feuchtigkeit verursacht Kurzschluss in elektrischen Schaltkreisen und beschädigt das Irinos-Messsystem.

Um Beschädigungen zu vermeiden, gehen Sie wie folgt vor:

- Lagern Sie die Irinos-Komponenten bzw. das Irinos-Messsystem trocken.
- Gleichen Sie die Irinos-Komponenten bzw. das Irinos-Messsystem vor Inbetriebnahme der Raumtemperatur an.
- Setzen Sie die Irinos-Komponenten bzw. das Irinos-Messsystem nicht der direkten Wärmestrahlung eines Heizgeräts aus.
- Bei Betauung schalten Sie das Irinos-Messsystem erst nach kompletter Trocknung ein oder nach einer Wartezeit von ca. 8 Stunden.

Achtung**Umgebungsbedingungen und chemische Beständigkeit**

Umgebungen, die für die Irinos-Komponenten nicht geeignet sind, stören den Betrieb. Chemische Mittel (z.B. Reinigungs- oder Betriebsmittel) können die Farbe, Form, Struktur der Geräteoberfläche verändern.

Die Irinos-Komponenten können beschädigt werden. Fehlfunktionen können die Folge sein.

Beachten Sie daher die folgenden Vorsichtsmaßnahmen:

- Betreiben Sie das Irinos-Messsystem nur in geschlossenen Räumen. Bei Zuwiderhandlung erlischt die Gewährleistung.
- Betreiben Sie das Irinos-Messsystem nur entsprechend den Umgebungsbedingungen, die in den technischen Daten angegeben sind.
- Schützen Sie das Irinos-Messsystem vor Staub, Feuchtigkeit und Hitze.
- Setzen Sie das Irinos-Messsystem keiner direkten Bestrahlung durch Sonnenlicht oder andere starke Lichtquellen aus.
- Ohne Zusatzschutzmaßnahmen, z.B. durch Zuführung sauberer Luft, kann das Irinos-Messsystem nicht an Orten mit erschwerten Betriebsbedingungen durch ätzende Dämpfe oder Gase eingesetzt werden.
- Verwenden Sie nur geeignete Reinigungsmittel.

Unzulässige und ungeeignete Reinigungsmittel können das Gerät beschädigen.

Verwenden Sie als Reinigungsmittel bei leichten Verschmutzungen nur Spülmittel, bei schweren Verschmutzungen Spiritus oder vergleichbare Alkohole. Verwenden Sie folgende Reinigungsmittel nicht:

- Aggressive Lösungs- oder Scheuermittel
- Dampfstrahler
- Druckluft
- Staubsauger

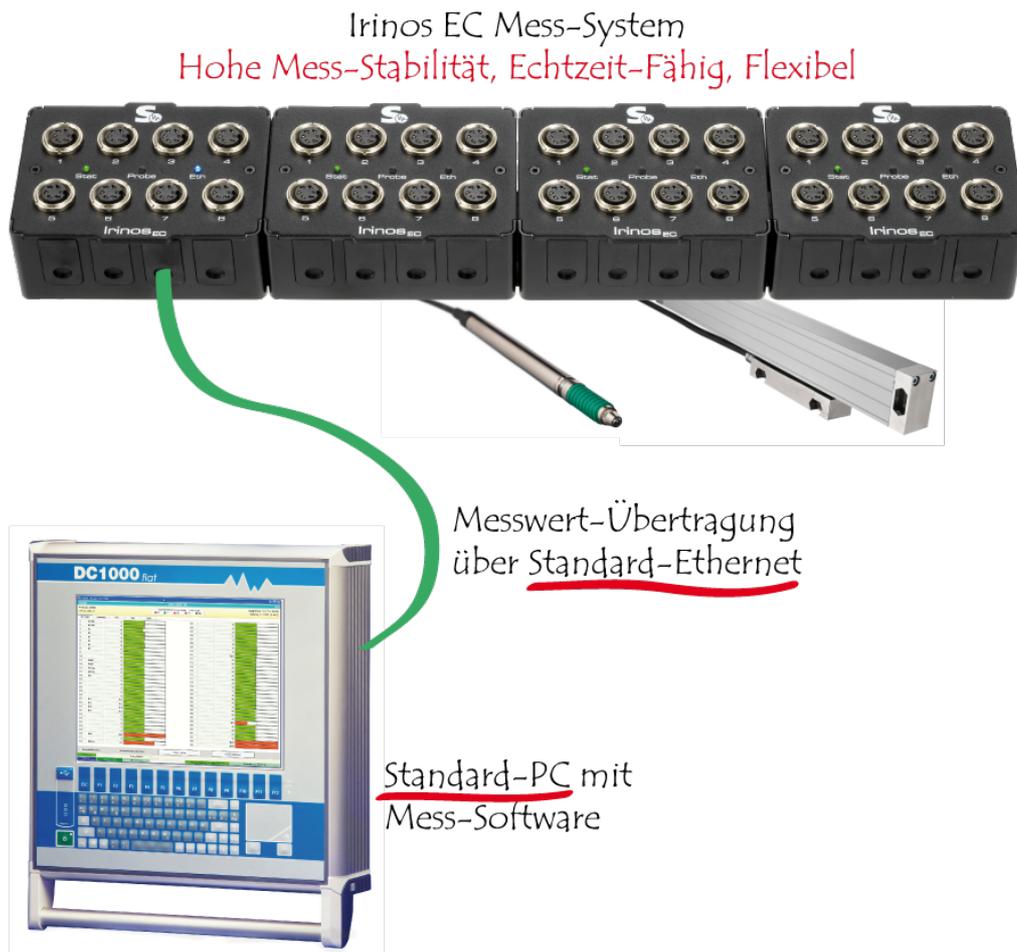
	Vorsicht
	Unbeabsichtigte Reaktion beim Reinigen des Irinos-Messsystem Wenn das Irinos-Messsystem beim Reinigen eingeschaltet ist, können Bedienelemente unbeabsichtigt ausgelöst werden. Das Irinos-Messsystem oder damit verbundene Komponenten können unbeabsichtigt reagieren. Personenschaden oder Maschinenschaden kann die Folge sein. Schalten Sie das Gerät vor der Reinigung aus.

2.3 System-Übersicht

Das Irinos-System übernimmt innerhalb eines industriellen Messsystems die hardwarenahen und zeitkritischen Funktionen.

Es ermöglicht den einfachen Anschluss verschiedener Messaufnehmer, z.B. induktive Messtaster. Über einen direkt anschließbaren Fuß- oder Handtaster sowie über digitale Ein-/Ausgänge lassen sich auch Steuerungsfunktionen integrieren.

Die Messwerte und E/A-Daten werden in normierter Form über Standard-Ethernet zum PC übertragen und dort in der Mess-Software weiterverarbeitet:



Für zeitkritische Anwendungen ist eine synchrone Messwertaufnahme in Echtzeit möglich. Die Messwerte werden dazu vom Irinos-System zeitgleich erfasst, zwischengepuffert und in sortierter Form an den PC weitergegeben. Eine wesentliche Eigenschaft ist dabei die Unabhängigkeit vom Zeitverhalten des PCs. Rundlauf- und Formmessungen sind somit selbst auf Laptop-PCs ohne spezielle Hardware bzw. Echtzeit-Funktion möglich. Die [Anbindung an die Messrechner-Software](#) erfolgt über eine Windows DLL, die sich über die Windows API ansprechen lässt. Die DLL ist mit allen aktuellen Windows-Versionen lauffähig (z.B. Windows 10).

Messrechner und Mess-Software sind als Gesamtsystem mit den Irinos-Komponenten von Messtechnik Sachs erhältlich. Aufgrund des offenen Konzepts kann das Irinos-System aber auch mit Messrechnern verschiedener anderer Hersteller kombiniert werden. Alternativ kann die DLL auch in eigene Mess-Software eingebunden werden.

2.3.1 Modularität

Das Irinos-System ist flexibel für verschiedene Mess-Aufgaben einsetzbar.

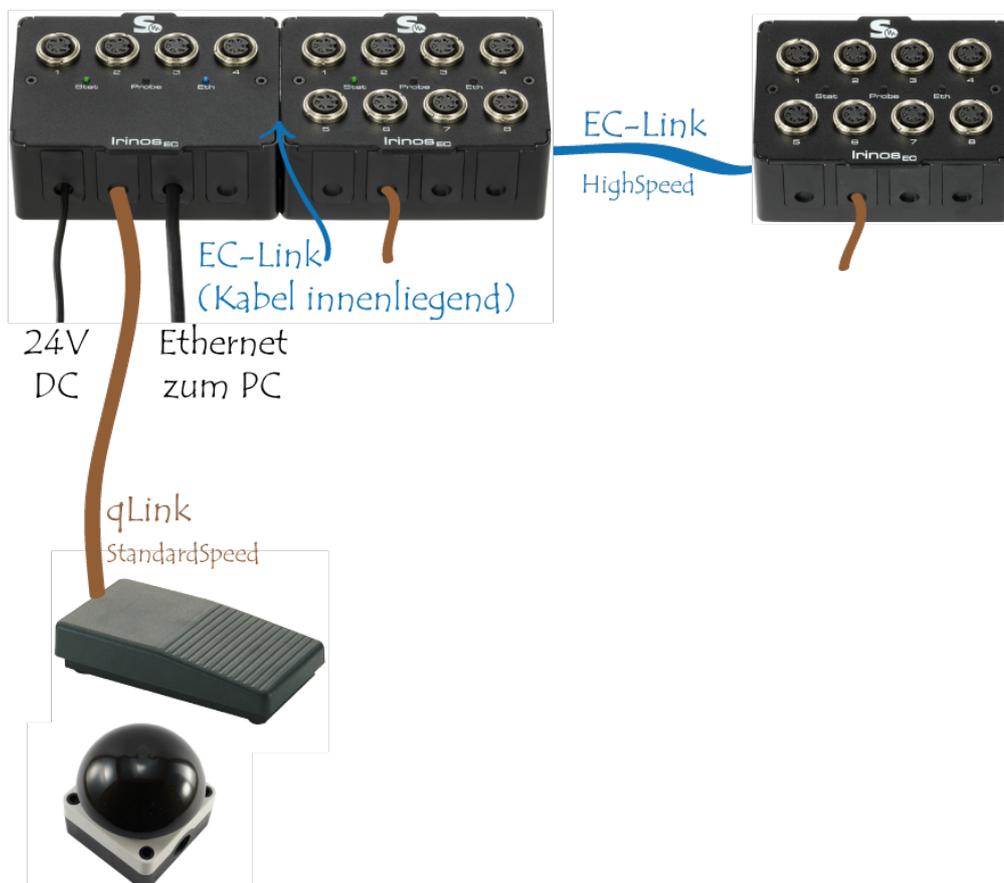
Ein Irinos-System besteht aus mindestens einer und maximal 8 Irinos-Boxen. Jede Irinos-Box hat eine feste Anzahl an Messeingängen (z.B. 2, 4 oder 8). Die Irinos-Boxen werden in Linien-Topologie über die EC-Link Schnittstelle

miteinander verbunden, d.h. zwischen zwei Irinos-Boxen gibt es immer ein EC-Link Kabel.

Werden 2 Boxen direkt nebeneinander platziert, so kann das EC-Link Kabel innen liegend platziert werden, d.h. es ist unsichtbar. Längere Abstände lassen sich durch externe Kabel überbrücken, wobei der Gesamtabstand zwischen der ersten und der letzten Irinos-Box 10m betragen darf.

Die EC-Link Schnittstelle vereint drei Aufgaben in einem:

- Datenaustausch zwischen den Irinos-Boxen.
- Zeit-Synchronisation aller Messkanäle.
- Spannungsversorgung der Irinos-Boxen.



Es können Irinos-Boxen mit verschiedenen Mess-Eingängen beliebig kombiniert werden.

Für den PC bzw. die Messrechner-Software spielt die Anzahl der Irinos-Boxen keine Rolle. Er sieht immer ein Irinos-System, dessen Messkanal-Anzahl sich

aus der Anzahl der im System vorhandenen Irinos-Boxen ableitet. Die Messwerte werden über die EC-Link Schnittstelle automatisch zwischen den Irinos-Boxen übertragen und zusammengefasst. Dann können sie über die Ethernet-Schnittstelle zusammenhängend ausgelesen werden.

Zusätzlich stellt jede Irinos-Box eine "qLink"-Schnittstelle bereit, über die simple Erweiterungen angeschlossen werden können. So ermöglicht die qLink-Schnittstelle beispielsweise den direkten Anschluss eines Fuß- oder Handtaster.

2.3.2 Synchronisation und Geschwindigkeit

Die einfachste Form der Messung ist die sogenannte **statische Messung**, bei der die Messwerte aller Messkanäle asynchron mit einer Update-Rate von ca. 30 Hz aktualisiert werden. Sie ist immer aktiv und bedarf keiner Parametrierung. Für zahlreiche Messaufgaben ist diese vollkommen ausreichend.

Neben der statischen Messwertaufnahme bietet das Irinos-System auch die Möglichkeit der schnellen **Echtzeit-Messwertaufnahme** bis 4000 Messwerte/s. Alle Messkanäle, die über die EC-Link Schnittstelle angebunden sind, können hierbei mit voller Geschwindigkeit einbezogen werden. Zudem können alle digitalen Ein- und Ausgänge einbezogen werden.



Die Echtzeit-Messung erfordert die Anbindung des Irinos-Systems an die Mess-Software über die [NmxDLL](#)^[62].

Alle Irinos-Boxen an der EC-Link Schnittstelle haben dieselbe System-Zeit, genannt „EC-Link-Zeit“ (Einheit: μs). Technisch bedingte Abweichungen der einzelnen Irinos-Boxen von der EC-Link-Zeit werden kontinuierlich korrigiert. Dies muss durch den Anwender weder parametriert noch kontrolliert werden. Die Abweichungen liegen in der Praxis im Bereich von wenigen MikroSekunden (μs).

Die Messwertaufnahme wird bei allen Irinos-Boxen basierend auf der EC-Link-Zeit gleichzeitig ausgelöst. Die Messwerte werden dann im internen Puffer der Irinos-Box zwischengespeichert und anschließend von der Master-Box mit den anderen Messwerten zusammengeführt. Dadurch können die Messwerte aller Kanäle gleichzeitig mit sehr hoher Geschwindigkeit erfasst werden.

Für eine **zeitliche begrenzte Echtzeit-Messung** gilt: Die Echtzeit-Fähigkeit ist unabhängig von der Anzahl der Messtaster, da jede Irinos-Box einen

eigenen Messwert-Puffer hat. So können beispielsweise alle Messkanäle gleichzeitig mit einer Messrate von 4000 Messwerten/s aufgezeichnet werden. Der Speicher ist so dimensioniert, dass bei maximaler Messrate die Messwerte über mindestens 10 Sekunden aufgezeichnet werden können. Bei geringerer Messrate erhöht sich diese Zeit entsprechend.

Lediglich die Übertragungszeit zum PC ist abhängig von Kanalzahl und Messrate. Als grober Richtwert können je nach Datentyp der angeschlossenen Messtaster zwischen 80.000 und 200.000 Messwerte/s übertragen werden.

Für eine **endlose Echtzeit-Messung** gilt: Die maximale Messrate ist abhängig von der Kanalzahl. Folgende maximalen Messraten werden empfohlen:

Empfohlene maximale Messrate (bei Endlos-Messung)	Anzahl Messkanäle
4000	1-16
2000	17-32
1000	33-64

2.3.3 Master vs. Slave

Jedes Irinos-System verfügt über genau eine Master-Box. Dies ist diejenige Irinos-Box, die über Ethernet an den PC angeschlossen ist. Alle anderen (optionalen) Irinos-Boxen werden als Slave-Box bezeichnet.

Nach dem Einschalten agiert jede Irinos-Box zunächst einmal als Slave-Box. Sie überprüft dann, ob ein Netzkabel angeschlossen ist (d.h. Ethernet-Link aktiv). Ist dies der Fall, wird sie automatisch zur Master-Box und teilt dies den anderen Boxen mit.

Die Verwendung von mehreren Ethernet-Verbindungen in einem System ist nicht zulässig.



Information für Anwender, die das "Irinos IR" - System bereits kennen:

Das "Irinos EC" - System unterscheidet sich in diesem Punkt vom klassischen "Irinos IR" - System, wo jede Box im Auslieferungszustand entweder eine Master- oder Slave-Box ist.

2.3.4 Spannungsversorgung

Beachten Sie unbedingt die [Sicherheitshinweise](#)^[21] in Bezug auf die elektrische Spannung! Ins besonders ist darauf zu achten, dass ein Netzteil mit Funktionskleinspannung (PELV) verwendet wird.

Alle Irinos-Boxen werden über eine gemeinsame 24 V - Spannung versorgt. Diese kann an beliebiger Stelle im System an einer dafür vorgesehenen [Buchse bzw. Klemme eingespeist werden](#)^[46]. Über die EC-Link- sowie qLink-Kabel wird Sie über das gesamte System hinweg verteilt.

Eine Ausnahme bilden die digitalen Ein-/Ausgänge, die je nach Box-Typ separat eingespeist werden müssen. Dies ermöglicht auch die Einbeziehung der digitalen Ein-/Ausgänge in einen Not-Aus-Kreis, ohne das Irinos-System komplett abzuschalten.

Die für die Messung sowie die Kommunikation benötigten Spannungen werden intern über galvanisch getrennte DC/DC-Wandler und erforderlichenfalls nachgeschaltete Linearregler erzeugt. Dies bedeutet, dass die internen Spannungen mehrerer Irinos-Boxen komplett voneinander getrennt sind. Die Störimmunität wird dadurch erhöht. Dies ist eine Voraussetzung für präzise Messergebnisse. Darüber hinaus wird die Entstehung von Masseschleifen damit verhindert.

Bitte beachten Sie dennoch, dass eine stabile und störungsarme 24 V – Versorgung Voraussetzung für einen einwandfreien Betrieb des Irinos-Systems ist. Verwenden Sie deshalb ein separates Netzteil mit Funktionskleinspannung (PELV) für das Irinos-System.

2.4 Produktbeschreibungen

Die Produktbeschreibungen geben eine Übersicht über die einzelnen Irinos-Boxen sowie verschiedenes Zubehör. Die Steckerbelegung sowie technische Details zu den einzelnen Anschlüssen finden Sie im Kapitel [Steckerbelegungen](#)^[39].

Verfügbare Messboxen (kaskadierbar über EC-Link - Schnittstelle)

Artikel-Nummer	Bezeichnung	Anzahl Messkanäle bzw. Ein-/Ausgänge	Kurzbeschreibung
828-6050	EC-TFV ³⁷ -8-TESA-M16-EPI	8	Messbox für induktive Wegaufnehmer Typ TESA Halbrücke oder kompatibel
828-6051	EC-TFV ³⁷ -4-TESA-M16-EPI	4	

Zubehör EC-Link Verbindungskabel

Artikel-Nummer	Länge	Kurzbeschreibung
828-6200	ca. 0,15m für innenliegende Verkabelung	EC-Link Verbindungskabel zur Kaskadierung von mehreren Irinos-Boxen über die EC-Link - Schnittstelle ²⁷
828-6201	0,5m	
828-6202	1,5m	
828-6203	3m	

Zubehör Tischnetzteile

Artikel-Nummer	Mitgeliefertes Anschlusskabel	Leistung
828-6130	Für EU: Schuko-Stecker, Typ F	Eingang: 115/230V, 50/60Hz

828-6131	Für USA/Mexiko: NEMA 5 Stecker, Typ B	Ausgang: 24V DC, max. 60W
828-6132	Für Indien: Stecker BS546, Typ D	
828-6133	Für Schweiz: Stecker SEV1011, Typ J	
828-6134	Für Großbritannien: Stecker BS1363, Typ G	

Zubehör Befestigung

Artikel-Nummer	Bezeichnung	Kurzbeschreibung
828-6150	EC-MHRM-1	Adapter zur Befestigung einer Irinos EC - Box an einer Hutschiene / Tragschiene / DIN-Rail / H-Rail

Zubehör Netzkabel

Artikel-Nummer	Länge	Kurzbeschreibung
828-6181	2m	Irinos EC Netzkabel, 2 x RJ45
828-6182	5m	
828-6183	10m	

828-6184	15m	
----------	-----	--

2.4.1 Grundaufbau Irinos-EC - Box mit EC-Link - Schnittstelle

Eine Irinos-EC - Box mit EC-Link - Schnittstelle besteht aus 2 Teilen, der Grundbox und dem Messmodul:



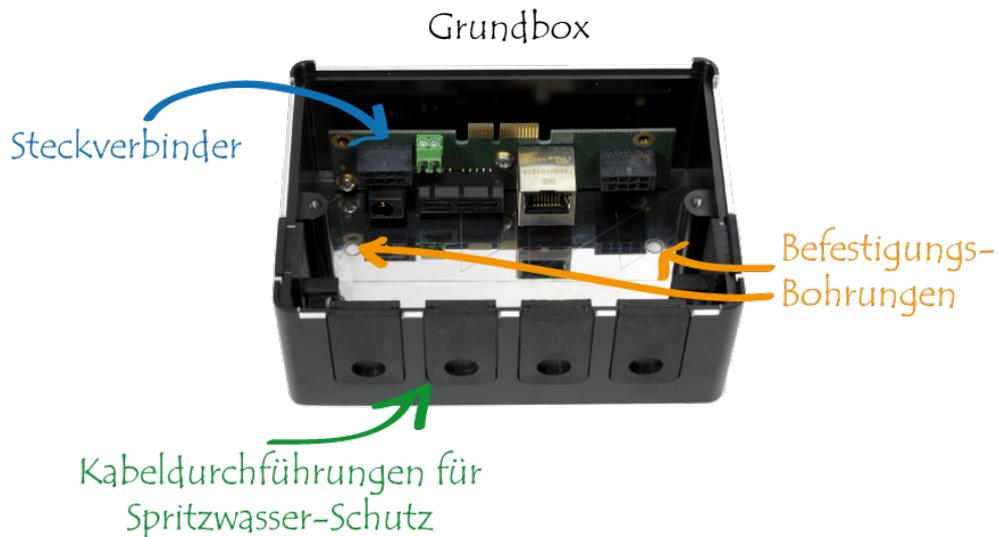
Grundbox

Die Grundbox ist für alle Irinos EC - Boxen identisch. Sie enthält keine aktive Elektronik.

Eine Besonderheit des Gehäusekonzeptes ist, dass alle Anschlüsse innen liegend sind (ausgenommen Messeingänge und digitale Ein-/Ausgänge). Dieses bringt mehrere Vorteile mit sich:

- Optisch ansprechend, da die Steckverbinder von außen nicht sichtbar sind.
- Verwendung von günstigen Standardkabeln möglich, keine Industrie-Steckverbinder erforderlich. Durch passende Kabeldurchführungen wird dennoch die erforderliche Dichtigkeit erreicht.

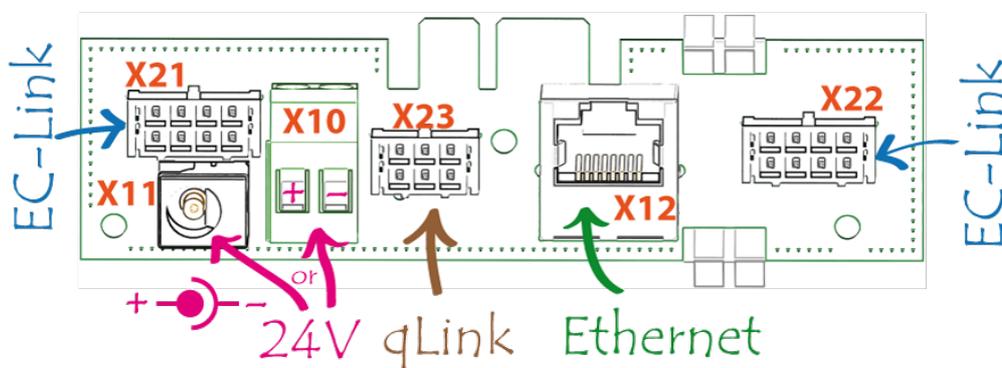
- Kompakter und dennoch sehr flexibler Systemaufbau möglich.
- Schneller Tausch des Messmoduls möglich.



Boden-seitig befinden sich 2 Bohrungen für Schrauben M4. Damit kann die Grundbox direkt befestigt werden. Alternativ kann hierüber auch der optionale Hutschienen-Adapter befestigt werden.

Die Grundbox enthält **Anschlüsse** für:

- X10 / X11: [Spannungsversorgung](#) ³¹ 24V DC (Eingang)
- X12: Ethernet-Schnittstelle zum Anschluss an den PC
- X21 / X22: EC-Link - Schnittstelle zur Kaskadierung mehrerer Irinos-Boxen
- X23: qLink - Schnittstelle zum Anschluss einfacher Erweiterungen (z.B. Fußtaster)





Die Leiterplatte mit den Steckverbindern kann sich geringfügig bewegen. Dies ist beabsichtigt!

Messmodul

Das Messmodul wird in die Grundbox montiert und enthält nach außen hin die Anschlüsse für die eigentliche Mess- oder Steuerungs-Funktion, wie z.B. die Anschlüsse für induktive Messtaster. Es enthält die komplette Elektronik.

Messmodul



Vorderansicht



Rückansicht

Jedes Messmodul hat frontseitig mindestens 2 LEDs, eine für den Box-Status und eine für die Ethernet-Verbindung.

LED "Stat" / grün: Anzeige des Box-Status	
Blinkend: lange an, kurz aus	Alles ok
Schnell blinkend	Es ist ein Fehler ^[67] aufgetreten. Der Fehler-Typ kann beispielsweise mit dem ITool ^[54] oder über die DLL-Schnittstelle ^[60] ausgelesen werden.

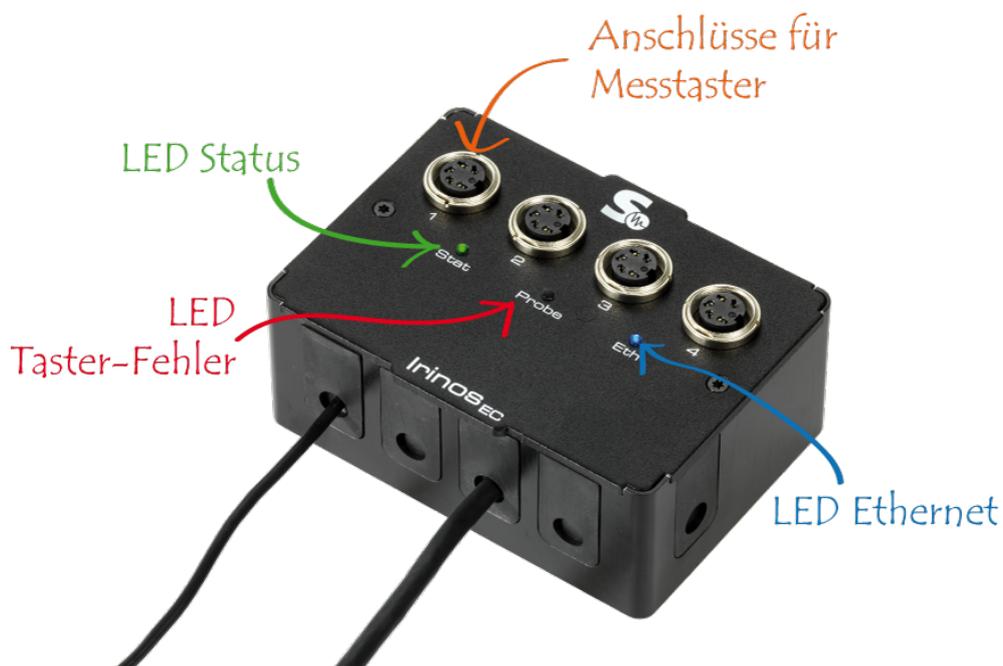
LED "Eth" / blau: Anzeige des Netzwerk-Status	
Aus	Es besteht keine Netzwerk-Verbindung. Dies ist bei Slave-Boxen ^[30] der Normalfall.

An	Eine Netzwerkverbindung besteht, jedoch werden keine Daten übertragen. Dies ist der Normalfall, wenn die physikalische Verbindung zum PC aufgebaut ist, jedoch noch keine Mess-Software gestartet wurde.
Blinkend	Eine Netzwerkverbindung besteht und es werden Daten übertragen. Dies ist der Normalfall, wenn die Verbindung zwischen Mess-Software und Irinos-System aufgebaut wurde.

2.4.2 EC-TFV für induktive Wegaufnehmer / Messtaster

Die Messbox EC-TFV ist für den Anschluss von induktiven Wegaufnehmern / Messtastern geeignet. Der unterstützte Messtaster-Typ kann der Beschriftung auf dem Typenschild entnommen werden.

Auf der Vorderseite der Mess-Box befinden sich die Anschlüsse für die induktiven Messtaster.



Zusätzlich zu den [Standard LEDs](#)^[36], hat diese Box eine "Probe"-LED:

LED "Probe" / rot : Anzeige eines Taster-Fehlers	
Aus	Alles ok oder kein Fehler erkennbar
An	Kurzschluss der Taster-Spannungsversorgung, d.h. Kurzschluss des Sinus-Erreger - Signals

Messwert-Erfassung

Alle Messeingänge werden **synchron abgetastet** (kein Multiplexing). Das Messverfahren berücksichtigt das vollständige Messsignal (d.h. integrierende Messung). Dieses Verfahren hat im Vergleich zur häufig eingesetzten 1-Punkt- oder 2-Punkt-Abtastung den Vorteil, dass seine Störempfindlichkeit viel besser ist. Der bei diesem Messverfahren eingesetzte analoge Filter ist auf die mechanischen Eigenschaften des jeweiligen Messtasters ausgelegt.

Technisch bedingt haben verschiedene Eingangskanäle nie exakt denselben Messwert bei identischem Eingangssignal. Um einen problemlosen Wechsel des Eingangs-Kanals bzw. einen Tausch der Irinos-Box zu ermöglichen, werden die Messeingänge deshalb werksseitig digital vorabgeglichen. Der Abgleich erfolgt so, dass maximaler negativer Nennauslenkung der Messwert -32.000 geliefert wird, in der Messtaster-Mitte der Messwert 0 und bei maximal positiver Nennauslenkung der Messwert +32.000.

Der interne Datentyp ist "16 Bit signed integer".

Digitalwert	Taster-Auslenkung Tesa Halbbrücke GT21 Empfindlichkeit: 73,75mV/V/μm	Taster-Auslenkung Tesa Halbbrücke GT61 Empfindlichkeit: 29,5mV/V/μm
- 32.000	- 2000 μm	- 5000 μm
0	0 μm	0 μm
+32.000	+ 2000 μm	+ 5000 μm
+32.767	Taster-Fehler erkannt	

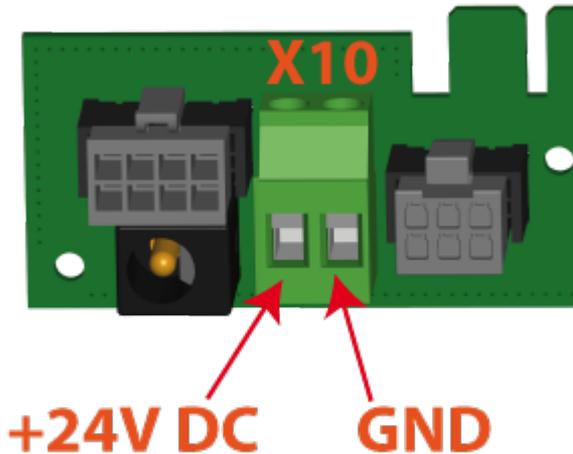
2.5 Steckerbelegungen

Achtung
<p>Für einen einwandfreien Betrieb sind stets fertig konfektionierte Kabel zu verwenden. Es besteht sonst die Gefahr, dass Komponenten des Irinos-Systems oder damit verbundene Komponenten beschädigt werden.</p> <p>Hiervon ausgenommen sind Klemmanschlüsse, z.B. für die Spannungsversorgung mit 24V DC über den Anschlussstecker X10³⁴.</p>

2.5.1 Spannungsversorgung 24V

Beachten Sie unbedingt die [Sicherheitshinweise](#)²¹ in Bezug auf die elektrische Spannung! Ins besonders ist darauf zu achten, dass ein Netzteil mit Funktionskleinspannung (PELV) verwendet wird.

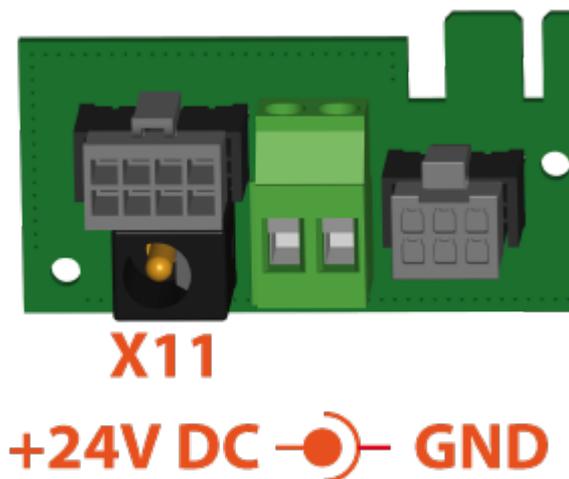
Anschlussmöglichkeit 1 über [Steckverbinder X10](#) 35 (Klemmanschluss):



Versehen Sie die Leitungsadern vor dem Anschluss mit Aderendhülse mit Kragen. Der maximale Leitungsquerschnitt ist $1,5\text{mm}^2$ / AWG16.

Um einen störungsfreien Betrieb auch unter äußerst ungünstigen Umgebungsbedingungen mit komplexen Verkabelungsverhältnissen gewährleisten zu können, ist eine geschirmte Leitung erforderlich. Verbinden Sie dazu den Leitungsschirm auf Seite der Irinos-Box mit dem FE-Bolzen der Bodenplatte. Dies ist innerhalb der Grundbox entsprechend markiert.

Anschlussmöglichkeit 2 über [Steckverbinder X11](#) 35 (Netzteil-Anschluss):



Der Anschluss erfolgt über einen 2,5mm Klinkenstecker, wie er für PELV-basierte Tisch-Netzteile de-facto Standard ist. GND ist außen-liegend, 24V ist innen-liegend.

2.5.2 Ethernet

Die Ethernet-Verbindung erfolgt über ein handelsübliches Patchkabel mit RJ45-Stecker nach EIA/TIA 568B. Verwenden Sie in jedem Fall ein geschirmtes Kabel nach Cat-6 oder besser.

Die Ethernet-Schnittstelle des Irinos-Systems hat eine „crossover“-Erkennung. Deshalb spielt es keine Rolle, ob ein Standard-Ethernet Kabel oder ein gekreuztes Kabel verwendet wird.

Die Datenrate beträgt 100 MBit/s.

Pin	Bezeichnung	Bemerkung
1	TX+	Transmission Data+
2	TX-	Transmission Data-
3	RX+	Receive Data+
4		unbenutzt
5		unbenutzt
6	RX-	Receive Data-
7		unbenutzt
8		unbenutzt

2.6 Montage

➔ Lesen Sie vor der Montage unbedingt die [Sicherheitshinweise](#)^[21]!

2.6.1 Lieferumfang / Prüfen der Lieferung

Prüfen der Lieferung

- Wenn Sie die Lieferung entgegen nehmen, prüfen Sie die Verpackung auf sichtbare Transportschäden.
- Wenn Transportschäden vorhanden sind, reklamieren Sie die Lieferung beim zuständigen Spediteur. Lassen Sie unverzüglich die Transportschäden durch den Spediteur bestätigen.
- Packen Sie die Irinos-Komponenten am Bestimmungsort aus.
- Bewahren Sie die Originalverpackung für einen erneuten Transport auf.
- Prüfen Sie den Verpackungsinhalt und Ihre extra bestellten Zubehörteile auf Vollständigkeit und Beschädigungen. Wenn der Verpackungsinhalt unvollständig oder beschädigt ist oder nicht Ihrer Bestellung entspricht, informieren Sie unverzüglich den Lieferanten.
- Bewahren Sie auch die mitgelieferten Unterlagen auf. Sie gehören zum Irinos-System.

Lieferumfang Irinos-Box

- 1 Grundbox mit Anschlusssteckern
- 1 Mess- oder E/A-Modul
- 5 Blind-Kabeldurchführungen zur Abdeckung nicht benötigter Gehäuse-Öffnungen
- 1 Kabeldurchführung für ein Tisch-Netzteil
- 1 Kabeldurchführung für ein handelsübliches Netzkabel
- 2 Unterlegscheiben Kunststoff für die Befestigung
- Begleitblatt mit Sicherheitshinweisen
- Warnhinweis DHCP-Server

Lieferumfang Zubehör

Artikel	Lieferumfang
Tisch-Netzteil	<ul style="list-style-type: none"> • Netzteil • Netz-Anschlusskabel entsprechend der Beschreibung im Artikel
Ethernet-Kabel	<ul style="list-style-type: none"> • Ethernet-Kabel
EC-MHRM-1 Hutschienen-Adapter	<ul style="list-style-type: none"> • Hutschienen-Adapter • 2 passende Inbus-Schrauben M4x6 zur Befestigung des Adapters an der Irinos-Box
EC-Link - Verbindungskabel	Verbindungskabel mit passenden Kabeltüllen
qLink - Verbindungskabel	Verbindungskabel mit passenden Kabeltüllen

2.6.2 Auswahl des Standorts

Die Irinos-Boxen sind als Feldgeräte sowohl für den Einsatz in geschlossenen Gehäusen, z.B. im Schaltschrank, als auch für eine Platzierung an oder in der Nähe der Messvorrichtung geeignet.

Insbesondere bei größeren Anlagen ist die Platzierung in der Nähe der Messvorrichtung zu bevorzugen. Sie bietet zwei wichtige Vorteile:

- Die Leitungen der Messtaster, Inkrementalgeber und sonstiger Sensoren können sehr kurz ausgeführt werden. Die Qualität der Analogsignale am Messeingang ist dadurch besonders gut. Zudem vereinfacht dies die Leitungsverlegung fern von möglichen Störquellen.
- Der Tausch eines Messtasters, z.B. bei einem Defekt, ist einfacher.

Berücksichtigen Sie für einen störungsfreien Betrieb des Irinos-Systems:
Platzieren Sie die Irinos-Boxen immer fern von möglichen Störquellen, wie z.B. Umrichtern oder Motorleitungen.

Ein wichtiger Faktor bei der Auswahl des geeigneten Standorts ist der Schutz gegen Staub und Wasser. Zwar sind die Irinos-Boxen so konzipiert, dass eine Dichtheit gegen übliche Verschmutzung gewährleistet ist. Damit dieser eingehalten werden kann, sind jedoch geeignete Steckverbinder der angeschlossenen Leitungen erforderlich. Die meisten der am Markt üblichen Messtaster und Inkrementalgeber haben jedoch Steckverbinder mit einer niedrigen Schutzart. Zur Erreichung der Schutzart müssten diese Steckverbinder getauscht werden.

Es ist deshalb empfehlenswert die Irinos-Boxen so zu platzieren, dass ein geringer Schutz ausreichend oder gar keine Schutz erforderlich ist.

Die Irinos-Boxen haben eine geringe Eigen-Wärmeentwicklung und sind für Industrie-übliche Umgebungstemperaturen ausgelegt. Zudem ist die integrierte Messelektronik besonders Temperatur-stabil.

Wählen Sie trotzdem einen Standort mit moderaten Umgebungstemperaturen. Der zulässige Temperaturbereich ist im jeweiligen Datenblatt angegeben. Vermeiden Sie insbesondere die Platzierung in der Nähe von Wärmequellen wie z.B. Kühlkörper anderer Geräte oder Heizelemente.

2.6.3 Befestigung

Irinos-Boxen lassen sich auf 2 verschiedene Wege befestigen:

1. Mittels 2 Schrauben M4, z.B. auf einer Blechplatte oder an einem Alu-Nutprofil (Item o.Ä.).
2. Mittels des Hutschiene-Adapters EC-MHRM-1 auf einer Hutschiene (auch bekannt als Tragschiene, DIN-Rail, H-Rail).

In jedem Fall ist ein sehr schneller Tausch einer Irinos-Box möglich.

Befestigung mit 2 Schrauben M4

Im Boden der [Grundbox](#)³⁴⁾ befinden sich 2 Löcher für Schrauben M4. Diese sind im Auslieferungszustand mit Gummipfropfen verschlossen.

Für die Befestigung werden benötigt:

- 2 Schrauben M4
- 2 Kunststoff-Unterlegscheiben M4 (im Lieferumfang enthalten)

- Geeigneter Schraubendreher

Die Vorgehensweise ist in folgendem Tutorial dargestellt:



Direkt-Montage

Befestigung mittels des Hutschienen-Adapters EC-MHRM-1

Der Hutschienen-Adapter wird mittels 2 Schrauben am Boden der [Grundbox](#)^[34] befestigt. Im Boden befinden sich hierfür 2 Löcher für Schrauben M4. Diese sind im Auslieferungszustand mit Gummipfropfen verschlossen.

Für die Befestigung werden benötigt:

- Hutschienen-Adapter EC-MHRM-1
- 2 Schrauben M4x6 (im Lieferumfang des Hutschienen-Adapters enthalten)
- 2 Kunststoff-Unterlegscheiben M4 (im Lieferumfang der Grundbox enthalten)
- Inbus-Schlüssel, Größe 2,5

Die Vorgehensweise ist in folgendem Tutorial dargestellt:



Hutschienen-Montage

2.6.4 Leitungen anschließen

Eine ordnungsgemäße Verkabelung ist entscheidend für den störungsfreien Betrieb des Irinos-Systems. Beachten Sie deshalb folgende Regeln:

- Verlegen Sie alle Leitungen räumlich getrennt von möglichen Störquellen, wie beispielsweise Umrichtern oder Motorleitungen.
- Vermeiden Sie unnötig lange Leitungen. Vermeiden Sie insbesondere „Kabelschleifen“.
- Alle Messleitungen sowie die EC-Link- und qLinkKabel müssen ausreichend geschirmt sein.
- Verwenden Sie die Verriegelungsmechanismen der Steckverbinder.
- Vermeiden Sie mechanische Belastungen, die auf die Leitungen einwirken können.



Achten Sie beim Einsatz in Schleppketten auf hierfür geeignete Leitungen.

Versehen Sie alle nicht verwendeten außenliegenden Steckverbinder mit einer Schutzkappe.



Die Leiterplatte mit den Steckverbindern in der [Grundbox](#)³⁴ kann sich geringfügig bewegen. Dies ist beabsichtigt!

Führen Sie zunächst die Montage der Irinos-Boxen durch. Schließen Sie dann die Leitungen in folgender Reihenfolge an:

1. Wenn mehr als 1 Irinos-Box verwendet wird: [EC-Link](#)^[47] - Leitungen anschließen.
2. Wenn über qLink anschließbares Zubehör verwendet wird: qLink - Leitungen anschließen.
3. [Ethernet](#)^[48]-Verbindung an der Master-Box anschließen.
4. [Spannungsversorgung](#)^[49] anschließen.

2.6.4.1 EC-Link Verkabelung



Die EC-Link - Verkabelung ist nur erforderlich, wenn 2 oder mehr Irinos-Boxen verwendet werden.

Grundlagen

- Die EC-Link - Schnittstelle ist ein Bussystem in Linientopologie. Jede Irinos-Box hat 2 [EC-Link - Steckverbinder](#)^[34].
- Es werden immer zwei Irinos-Boxen miteinander über eine EC-Link - Leitung verbunden. Bei der ersten und letzten Irinos-Box bleibt jeweils ein EC-Link - Steckverbinder unbenutzt.
- Eine manuelle Terminierung ist nicht erforderlich. Die notwendige Terminierung nimmt das Irinos-System beim Einschalten automatisch vor.
- Die maximal zulässige Gesamt-Länge der EC-Link - Verkabelung beträgt 10m.
- Ein Irinos-System darf aus maximal 8 Irinos-Boxen bestehen (inklusive Master-Box).

Vorgehensweise

Die Vorgehensweise bei der Durchführung der EC-Link - Verkabelung ist in folgendem Tutorial dargestellt:



EC-Link - Verkabelung

2.6.4.2 Ethernet anschließen



Bei einem System mit mehreren Irinos-Boxen, darf nur an einer Box die Ethernet-Verbindung hergestellt werden. Diese wird dann als [Master-Box](#)^[30] bezeichnet.

Bei der Ethernet-Schnittstelle des Irinos-Systems handelt es sich um eine Standard Ethernet-Schnittstelle, wie sie beispielsweise auch zur IT-Vernetzung verwendet wird. Das Irinos-System kann daher grundsätzlich auch mit Standard-Ethernet-Switches verwendet werden.

Die Datenkommunikation zwischen dem Irinos-System und dem PC ist fehlertolerant aufgebaut, so dass bei einem Paketverlust automatisch eine Übertragungswiederholung durchgeführt wird. Diese Wiederholung führt jedoch immer zu einer nennenswerten Verzögerung bei der Verfügbarkeit der Messwerte. Eine Übertragungswiederholung sollte deshalb in der Praxis eine Ausnahme sein.

Um die Anzahl der Übertragungswiederholungen zu minimieren, wird deshalb eine **direkte – Verbindung zwischen dem Irinos-System und dem PC dringend empfohlen**. Verbinden Sie dazu die Ethernet-Schnittstelle des Irinos-Systems mit einer freien Ethernet-Schnittstelle des PCs. Erfahrungsgemäß treten Übertragungswiederholungen dadurch praktisch nie auf.

Ein Betrieb des Irinos-Systems über Router, VPN-Verbindungen, Funk-Verbindungen (WLAN) oder Ähnlichem, ist nicht vorgesehen.

Die Ethernet-Schnittstelle des Irinos-Systems hat eine automatische „Cross-Over-Erkennung“. Es spielt somit keine Rolle, ob ein 1:1 Ethernet-Kabel oder ein gekreuztes Ethernet-Kabel verwendet wird.



Im Auslieferungszustand ist der DHCP-Server des Irinos-Systems aktiviert. Dies ist die ideale Einstellung für eine direkte Verbindung zum PC.

Bevor die Irinos-Box in einem IT-Netzwerk betrieben wird, muss der DHCP-Server über eine direkte Verbindung deaktiviert werden. Dies geschieht über das Irinos-Tool. Nähere Informationen dazu entnehmen Sie der Dokumentation des Irinos-Tools.

Vorgehensweise

Die Vorgehensweise bei der Durchführung der Ethernet - Verkabelung ist in folgendem Tutorial dargestellt:



Ethernet anschliessen

2.6.4.3 Spannungsversorgung anschließen

Beachten Sie unbedingt die [Sicherheitshinweise](#)^[21] in Bezug auf die elektrische Spannung! Ins besonders ist darauf zu achten, dass ein Netzteil mit Funktionskleinspannung (PELV) verwendet wird.

Beachten Sie die generellen Hinweise zur [Spannungsversorgung](#)^[31].

Es wird eine 24V DC - Spannungsversorgung benötigt.

Diese kann entweder über einen 2,5mm DC-Steckverbinder erfolgen (-> Tischnetzteil), oder über Klemmen.

Vorgehensweise / Tisch-Netzteil / Anschlussstecker X11

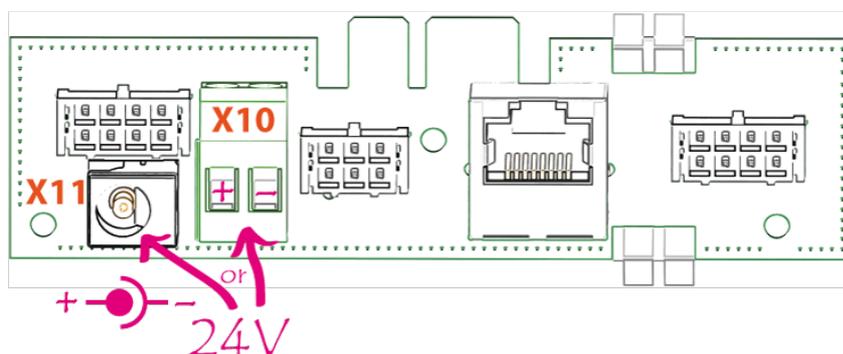
Die Vorgehensweise zum Anschluss der Spannungsversorgung über ein Tisch-Netzteil, ist in folgendem Tutorial dargestellt:



Netzteil anschliessen

Vorgehensweise / Klemmen X10

Wird kein Tisch-Netzteil, sondern beispielsweise ein Schaltschrank-Netzteil verwendet, so kann der 24V - Anschluss auch über die grünen Klemmen (X10) erfolgen. Die Klemmenbelegung ist im Aufkleber innerhalb der Irinos-Box dargestellt:



Verwenden Sie stets Aderendhülsen mit Kragen für die Leitungs-Adern. Die Klemmen sind für Leiterquerschnitte von 0,25mm² bis 1,5mm² ausgelegt.

Um auch unter äußerst ungünstigen Umgebungsbedingungen eine optimale EMV-Störfestigkeit entsprechend den Anforderungen der CE-Kennzeichnung

gewährleisten zu können, muss ein geschirmtes Kabel verwendet und an den Bolzen für Funktionserde angeschlossen werden. Dieser ist am Gehäuseboden angebracht (Gewinde M3).

2.6.5 Messmodule einbauen

Achtung

Beschädigung durch unsachgemäßen Einbau

Achten Sie beim Einbau des Messmoduls darauf, dass dieses nicht verkantet.

Stellen Sie sicher, dass alle Leitungen so angeschlossen sind, dass ausreichend Platz für das Mess-Modul vorhanden ist. Achten Sie insbesondere darauf, dass der Steckverbinder des Messmoduls auf die Stecker-Platine gesteckt werden kann.

Vorgehensweise

Die Vorgehensweise zum Einbau des Mess-Moduls (Fronteinheit), ist in folgendem Tutorial dargestellt:



Modul-Einbau

2.7 Inbetriebnahme und Kennenlernen

Das Irinos-System ist so konzipiert, dass keine Konfiguration des Systems oder der Komponenten erforderlich ist. Einzige Ausnahme hiervon sind die

Netzwerkeinstellungen (IP-Adresse), die über das [ITool](#)^[54] umkonfiguriert werden kann.

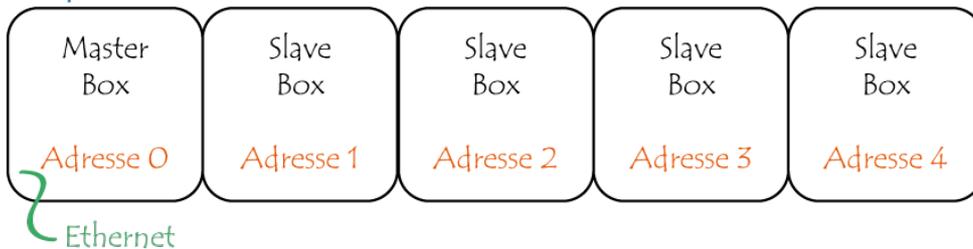
Sobald die [Montage und Verkabelung](#)^[41] durchgeführt wurde, kann das Irinos-System sofort eingeschaltet werden, indem das Netzteil mit Spannung versorgt wird.

2.7.1 Box-Adressierung

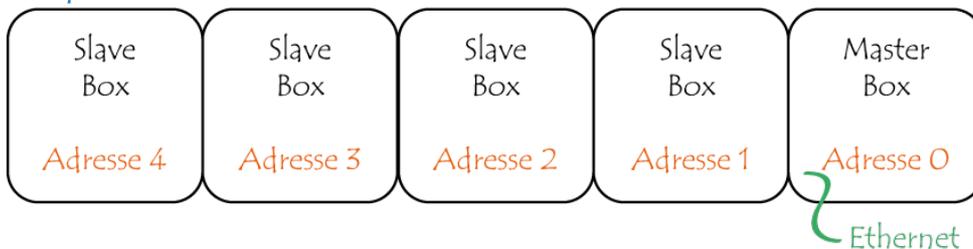
Alle Irinos-Boxen werden nach dem Einschalten automatisch adressiert. Die Master-Box hat immer die Adresse 0. Die Slave-Boxen werden fortlaufend in der Reihenfolge nummeriert, wie sie angeschlossen sind. Die Box-Reihenfolge ist auch ausschlaggebend für die initiale Nummerierung der Messeingänge bzw. der digitalen Ein-/Ausgänge.

Folgende Abbildung zeigt einige Beispiele für die Box-Adressierung:

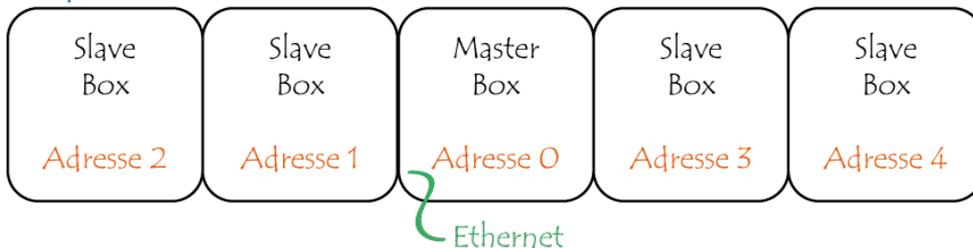
Beispiel 1



Beispiel 2



Beispiel 3



Um bei einer System-Erweiterung keine Verschiebung der Adressierung zu bekommen, wird empfohlen die erste oder die letzte Box als Master-Box zu verwenden (Beispiele 1 und 2).

Die **Dauer** des Adressierungs-Vorgangs ist abhängig von der Anzahl der angeschlossenen Boxen. Typischerweise dauert er nur wenige Sekunden.

Im Rahmen der Box-Adressierung werden auch die erste und die letzte Box ermittelt. Bei beiden wird automatisch die Bus-**Terminierung** aktiviert.

Im Rahmen der Inbetriebnahme sollte nach dem Einschalten geprüft werden, ob alle Boxen korrekt angeschlossen und damit korrekt adressiert werden. Dies geht beispielsweise über den [Webserver](#)^[55].

2.7.2 Netzwerk-Konfiguration

Eine Irinos-Box hat einen integrierten DHCP-Server. Dieser ist im Auslieferungszustand aktiviert. Die IP-Adresse des Irinos-Systems ist 192.168.3.99, die Subnetz-Maske 255.255.255.0. Sofern die Ethernet-Schnittstelle des PCs als „DHCP-Client“ konfiguriert ist, bekommt er beim Verbindungsaufbau automatisch eine IP-Adresse aus dem Adressbereich 192.168.3.100 bis 192.168.3.254 zugewiesen. Es ist dann keine Netzwerk-Konfiguration erforderlich. Es wird dennoch empfohlen PC-seitig eine feste Netzwerkadresse zu vergeben (z.B. 192.168.3.98).



Im Auslieferungszustand ist der DHCP-Server des Irinos-Systems aktiviert. Dies ist die ideale Einstellung für eine direkte Verbindung zum PC.

Bevor die Irinos-Box in einem IT-Netzwerk betrieben wird, muss der DHCP-Server über eine direkte Verbindung deaktiviert werden. Dies geschieht über das [Irinos-Tool](#)^[54]. Nähere Informationen dazu entnehmen Sie der Dokumentation des Irinos-Tools.

Falls die Nutzung der DHCP-Funktion nicht gewünscht ist, gibt es 2 Möglichkeiten:

- a) Der DHCP-Server an der Master-Box bleibt aktiviert. Der PC erhält aber eine feste IP-Konfiguration. Verwenden Sie hierzu beispielsweise folgende Netzwerkeinstellungen am PC:
IP-Adresse: 192.168.3.98
Subnetzmaske: 255.255.255.0
- b) Der DHCP-Server an der Master-Box wird über das [Irinos-Tool](#)^[54] deaktiviert. Die IP-Adresse der Irinos-Box kann frei vergeben werden. Der PC erhält eine feste IP-Konfiguration.
Die genaue Vorgehensweise hierzu entnehmen Sie der Dokumentation des Irinos-Tools.

Ob die Netzwerkverbindung funktioniert, kann am Einfachsten über den [Web-Server](#)^[55] des Irinos-Systems getestet werden. Öffnen Sie dazu einen Webbrowser und geben Sie in die Adresszeile die IP-Adresse des Irinos-Systems ein. Bei einer funktionierenden Netzwerkverbindung erscheint nun die Webseite mit der Messwertanzeige. Unter "[Erste Hilfe: Netzwerkverbindung](#)"^[78] ist die typische Vorgehensweise bei Verbindungsproblemen beschrieben.

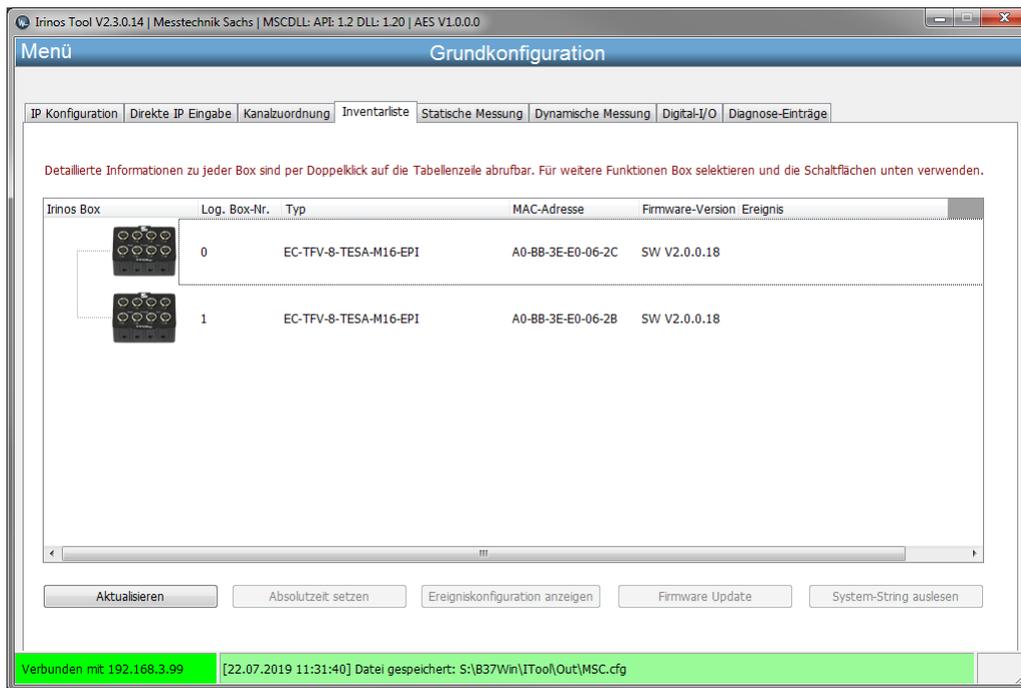
2.7.3 Irinos-Tool

Das Irinos-Tool ist ein Konfigurations- und Testwerkzeug für das Irinos-System. Zu seinem Funktionsumfang gehören:

- Ändern der Netzwerk-Konfiguration des Irinos-Systems.
- Übersicht der verfügbaren Messkanäle.
- Diagnose der Inkrementalgeber-Signale (1Vss).
- Anzeige der statischen Messwerte.
- Übersicht der an ein Irinos-System angeschlossenen Irinos-Boxen.
- Durchführen von Firmware-Updates.
- Auslesen und Speichern des Diagnose-Speicher – Inhaltes.

Weitere Informationen entnehmen Sie der separat verfügbaren Dokumentation des Irinos-Tools.

Es wird empfohlen das Irinos-Tool auf dem an das Irinos-System angeschlossenen Messrechner bereit zu halten, so dass es im Diagnose-Fall schnell als Hilfsmittel herangezogen werden kann. Für das Irinos-Tool fallen keine Lizenzgebühren an, solange es ausschließlich in Verbindung mit dem Irinos-System verwendet wird.



2.7.4 Web-Server

Das Irinos-System verfügt über einen integrierten Web-Server, der als Inbetriebnahme- und Diagnose-Hilfe dient. Der Zugriff auf den Web-Server erfolgt von einem Webbrowser wie beispielsweise InternetExplorer, Firefox, Chrome oder Opera. Geben Sie dazu die IP-Adresse des Irinos-Systems in die Adresszeile des Webbrowsers ein (Auslieferungszustand 192.168.3.99).



Die Darstellung der Webseiten wurde mit den Webbrowsern InternetExplorer 11, Firefox und Chrome erfolgreich getestet. Aufgrund der unterschiedlichen Interpretation von Standards, kann eine einwandfreie Funktion nicht mit allen Browsern und Browser-Versionen garantiert werden.

Es stehen 3 verschiedene Webseiten zur Verfügung:

Measurement (aktuelle Messwerte)

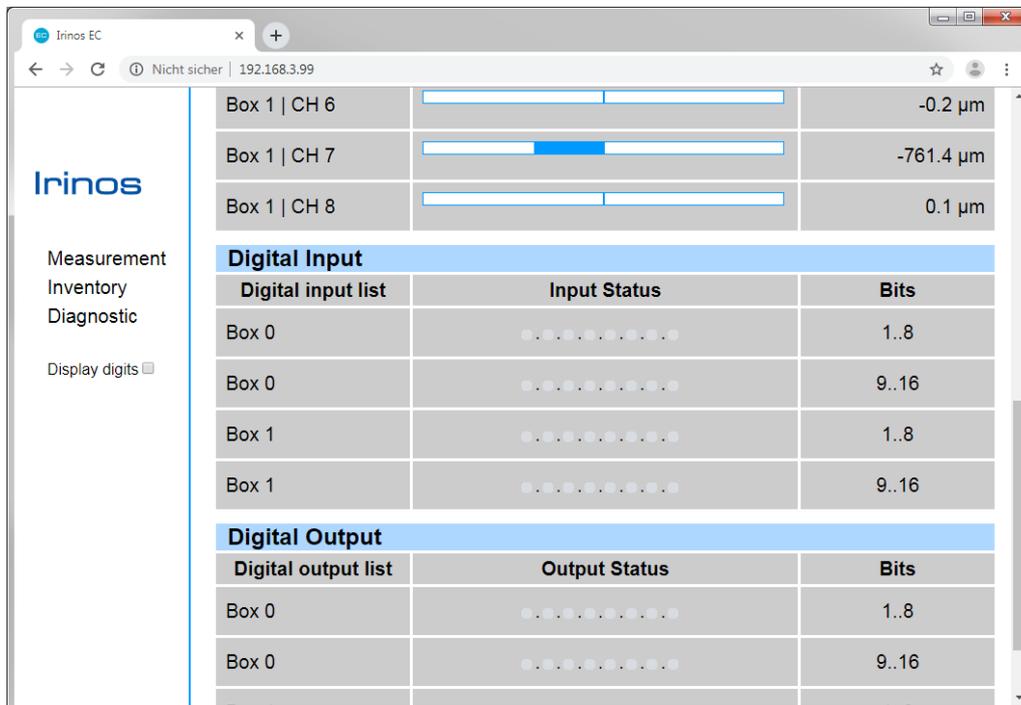
Auf der Webseite "Measurement" werden die aktuellen Messwerte der Messeingänge sowie der aktuelle Zustand der digitalen Ein-/Ausgänge live angezeigt (Updaterate ca. 4 Hz).

Damit können:

- Die Messtaster bereits zu einem Zeitpunkt eingestellt und getestet werden, an dem noch keine Messrechner-Software zur Verfügung steht.

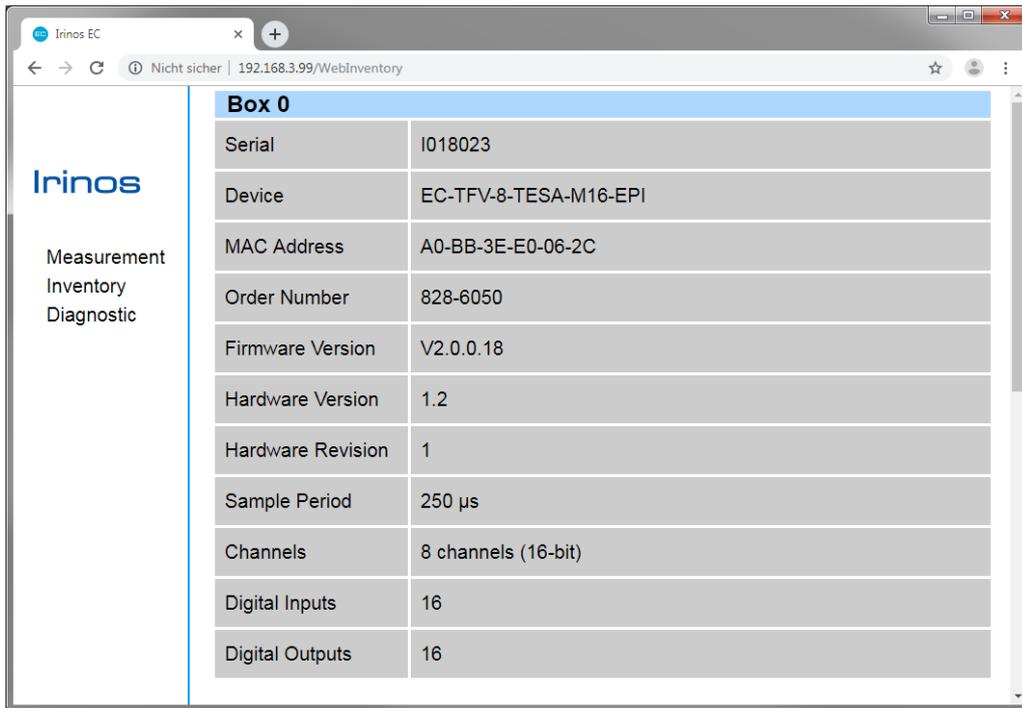
- Die vom Irinos-System gelieferten Messwerte mit denjenigen verglichen werden, welche die Messrechner-Software anzeigt.

Bitte beachten Sie bei induktiven Messtastern: Die angezeigte Einheit ist gültig für Standard-Taster. Bei Taster mit größerem Hub (z.B. ±5mm) und anderer Empfindlichkeit muss eine manuelle Umrechnung erfolgen.



Inventory (Box-Übersicht)

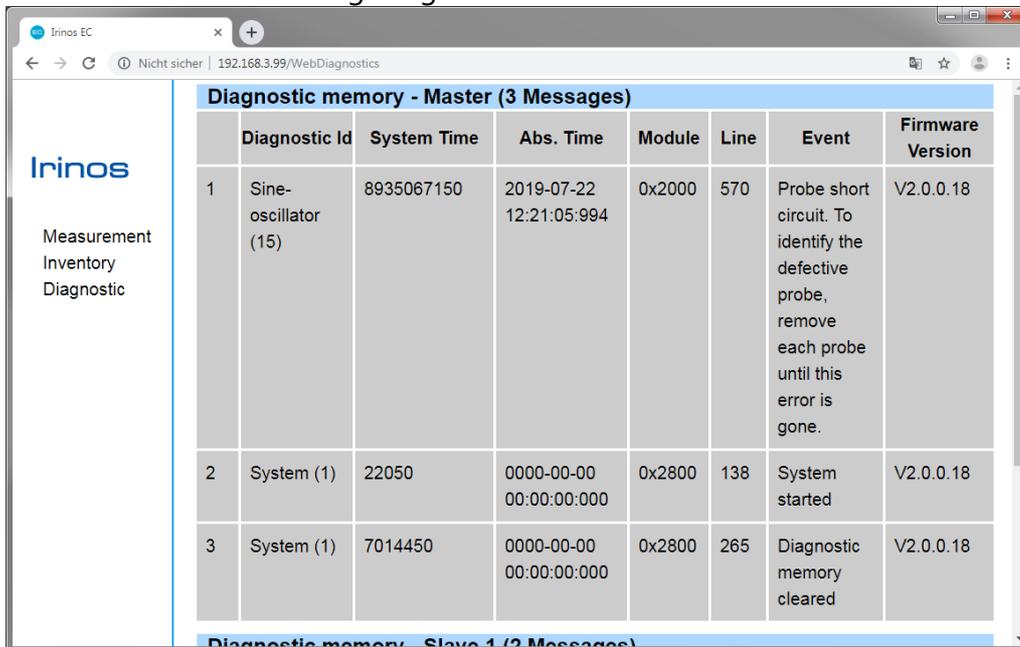
Auf der Webseite "Inventory" wird eine Übersicht aller im Irinos-System vorhanden Irinos-Boxen mit den wichtigsten Box-Informationen angezeigt:



Information	Beispiel	Beschreibung
Serial	I018023	Seriennummer der Irinos-Box
Device	EC-TFV-8-TESA-M16-EPI	Bezeichnung der Irinos-Box
MAC Address	A0-BB-3E-E0-06-2C	Eindeutige MAC-Adresse der Irinos-Box.
Order Number	828-6050	Bestellnummer. Diese Nummer beginnt immer mit 8.
Firmware Version	V2.0.0.18	Versionsnummer der Firmware.
Hardware Version	V1.2	Versionsnummer der Mess-Hardware.
Hardware Revision	1	Kompatibilitätskennung der Hardware für die Firmware.
Sample Period	250 μ s	Interne Abtastperiode in μ s.
Channels	8 channels (16-bit)	Anzahl der Messkanäle und interner Datentyp der jeweiligen Messkanäle.
Digital Inputs	16	Anzahl der an der Box verfügbaren digitalen Eingänge
Digital Outputs	16	Anzahl der an der Box verfügbaren digitalen Ausgänge

Diagnostic (Diagnose-Speicher)

Auf der Webseite Diagnostic wird der Inhalt des Diagnose-Speichers der einzelnen Irinos-Boxen angezeigt:



The screenshot shows a web browser window with the URL 192.168.3.99/WebDiagnostics. The page title is "Diagnostic memory - Master (3 Messages)". The table contains the following data:

	Diagnostic Id	System Time	Abs. Time	Module	Line	Event	Firmware Version
1	Sine-oscillator (15)	8935067150	2019-07-22 12:21:05:994	0x2000	570	Probe short circuit. To identify the defective probe, remove each probe until this error is gone.	V2.0.0.18
2	System (1)	22050	0000-00-00 00:00:00:000	0x2800	138	System started	V2.0.0.18
3	System (1)	7014450	0000-00-00 00:00:00:000	0x2800	265	Diagnostic memory cleared	V2.0.0.18

Below the table, the header for "Diagnostic memory - Slave 1 (2 Messages)" is visible.

Spalte	Beispiel	Bedeutung
Diagnostic-Id	Sine-oscillator (15)	Typ des Diagnose-Ereignisses ^[67] .
System Time	8935067150	Link-Zeit ^[29] in μs seit Einschalten des Irinos-Systems. (Diese interne Zeit ist für alle Irinos-Boxen eines Systems identisch.)
Abs. Time	2019-07-22 12:21:05:994	Datum und Uhrzeit, an dem das Ereignis aufgetreten ist. Jahr-Monat-Tag Stunde:Minute: Sekunde: Millisekunde
Module	0x2000	Zusatz-Informationen für den Support.
Line	570	
Event	Probe short circuit. To identify the defective probe, remove each probe until this error is gone.	Hilfs-Text für das Ereignis.
Firmware-Version	V2.0.0.18	Firmware-Version, mit welcher das Ereignis aufgetreten ist.

2.8 Software-Schnittstelle

Das Irinos EC - System bietet 3 verschiedene Möglichkeiten zur Integration in die Mess-Software, wobei die NmxDLL die bevorzugte ist:

	NmxDLL ⁶²	ASCII / Telnet ⁶³	MscDLL ⁶⁷
Typ	Windows-basierte DLL-Schnittstelle (ab Win7)	ASCII-basierte Messwertabfrage über Telnet oder UDP	Windows-basierte DLL-Schnittstelle (ab Win XP)
Statische Messung	Ja	Ja	Ja
Echtzeit-Messung	Ja	Nein	Nein
Digitale E/A-Daten austauschen	Ja	Ja	Ja
Auslesen von Status-Informationen (z.B. aktuelles Diagnose-Ereignis)	Ja	Nein	Ja
Parametrierung von Messkanälen (z.B. Inkrementalgeber Referenzmarke ein-/ausschalten)	Ja	Nein	Ja
Unterstützung von anderen Messsystemen	○ Irinos IR ab Firmware-Version 2	○ Irinos IR ab Firmware-Version 2	○ Irinos IR ab Firmware-Version 1 ○ Ältere Systeme

Hinweis	Bevorzugte Schnittstelle für Standard-Applikationen	Schnell zu implementierende Lösung für einfachste Anwendungen	Kompatibilität mit bestehender Integration der MscDLL in Mess-Software (Achtung: dynamische Messung nicht unterstützt!). Nicht für Neu-Integration empfohlen.
----------------	---	---	--

Die NmxDLL sowie die MscDLL sind in separatem Referenzhandbuch jeweils ausführlich beschrieben. Konsultieren Sie das jeweilige Referenzhandbuch für detaillierte Informationen.

Allen Software-Schnittstellen ist gemein, dass die Software-Implementierung unabhängig ist von der Anzahl der Messkanäle.

So spielt es für die Software-Implementierung beispielsweise keine Rolle, ob ein System 4 oder 64 Messkanäle hat. Es muss immer nur eine einzige Verbindung zum Irinos-System aufgebaut und verwaltet werden. Lediglich die Anzahl der verfügbaren Messkanäle hängt von der Kanalzahl ab.

2.8.1 NmxDLL Kurzübersicht

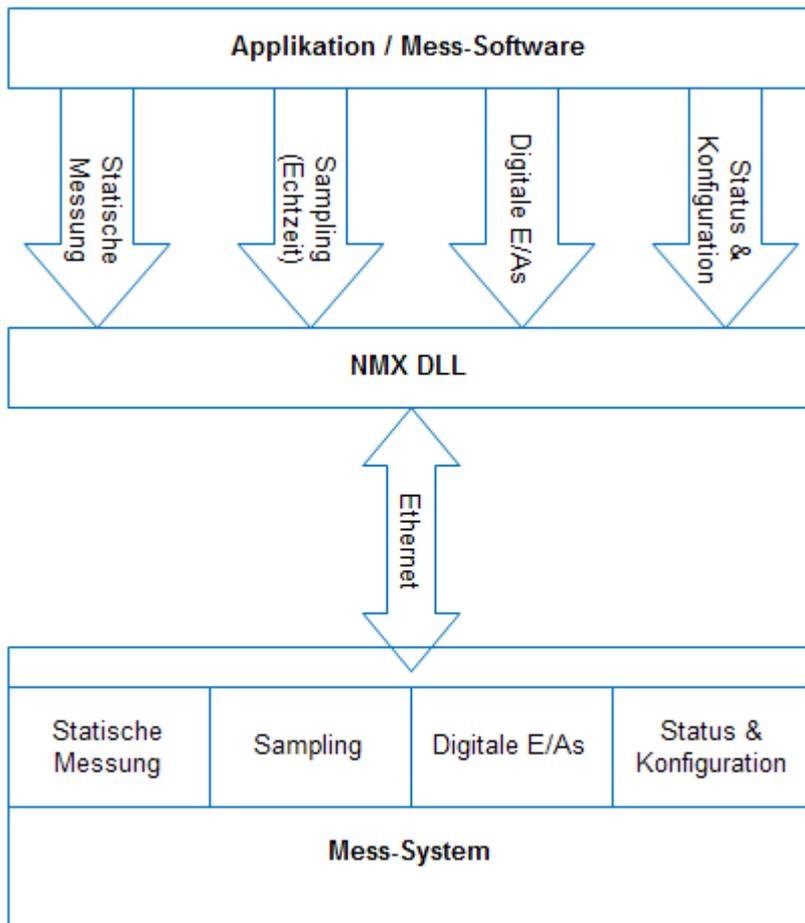


Dies ist lediglich eine Kurz-Übersicht!

Detaillierte Informationen finden Sie im Referenz-Handbuch der NmxDLL.

Die NmxDLL ist das Bindeglied zwischen der Mess-Software und dem Irinos-System. Sie bietet folgende Möglichkeiten:

- Auslesen der Messwerte vom Irinos-System (statisch und/oder in Echtzeit).
- Auslesen von Status-Informationen (z.B. aktuelles [Diagnose-Ereignis](#)^[67]).
- Lesen des Zustands der Digitalen Eingänge / Zustand digitaler Ausgänge setzen
- Parametrierung des Irinos-Systems



Für kleine Applikationen mit geringem Funktionsumfang genügt die Verwendung weniger Funktionsaufrufe der NmxDLL. Weitere ermöglichen auch die einfache Realisierung komplexer Applikationen.

Für die Integration in Mess-Software stehen verschiedene Beispiel-Programme zur Verfügung.

2.8.2 ASCII- / Telnet-Schnittstelle

Für einfachste Anwendungen bietet das Irinos EC eine ASCII-basierte Text-Schnittstelle. Dies ist vergleichbar mit der früher üblichen Anbindung von Mess-Hardware über RS232.

Der Zugriff auf die ASCII-Schnittstelle kann entweder über das TCP-basierte Telnet oder über UDP erfolgen. Während das ASCII-Protokoll selbst identisch ist, unterscheiden sich die beiden Möglichkeiten wie folgt:

	Telnet / TCP	UDP
--	---------------------	------------

Port-Nummer	TCP 22515	UDP 22515
Paketverlust	Bei Paketverlust automatische Wiederholung der Datenübertragung auf Protokoll-Ebene (TCP).	Bei Paketverlust auf UDP-Ebene ist eine manuelle Wiederholung des Datenaustauschs erforderlich.
Latenzzeit	Theoretisch sehr lange Latenzzeit möglich. Bei 1:1-Ethernet-Verbindung in der Praxis jedoch vernachlässigbar.	Sehr kurze Latenzzeiten möglich.
Max. Text-Länge	16000 Zeichen/Bytes	1450 Zeichen/Bytes bei 1:1-Ethernet-Verbindung

Die Daten werden immer im Frage -> Antwort - Verfahren ausgetauscht, d.h. die Mess-Software sendet eine Anfrage an das Irinos-System und dieses sendet eine Antwort. Jede Anfrage / Antwort besteht aus einem Art "Header", den eigentlichen Daten sowie einer Ende-Kennung.

Über den Header können anhand einer Kennung verschiedene Befehle unterschieden werden. Die Nachfolgenden Daten sind je nach Frage/Antwort optional. Die Ende-Kennung wird über "CarriageReturn (CR)" und "LineFeed (LF)" beendet und wird im Folgenden als {CRLF} dargestellt. Dabei hat CR den ASCII-Code 0x0D und LF den ASCII-Code 0x0A.

Anmerkung: Das Irinos-System erkennt auch nur CR oder nur LF als Ende, sendet in der Antwort aber immer beides zurück.

Das Protokoll wird anhand folgender Beispiele erläutert:

Befehl	Anfrage	Antwort	Anmerkungen
--------	---------	---------	-------------

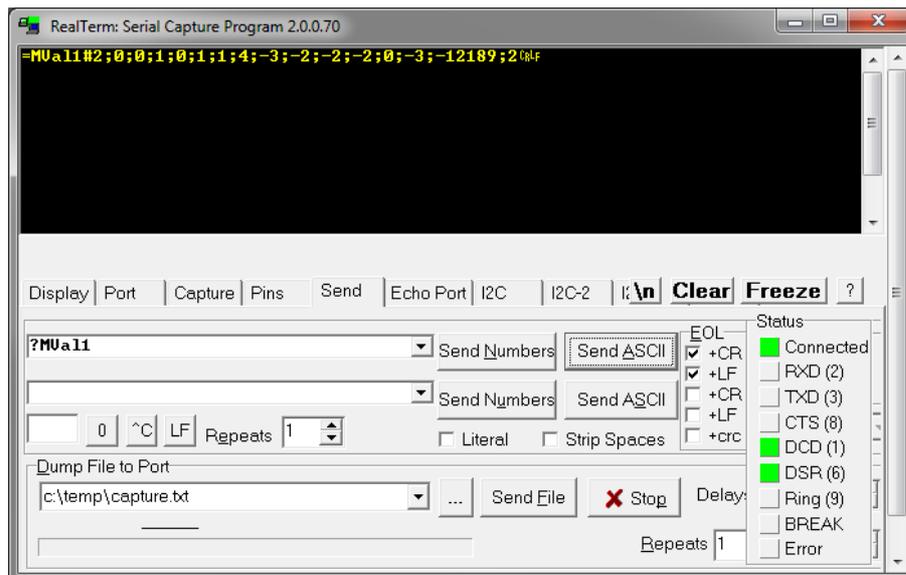
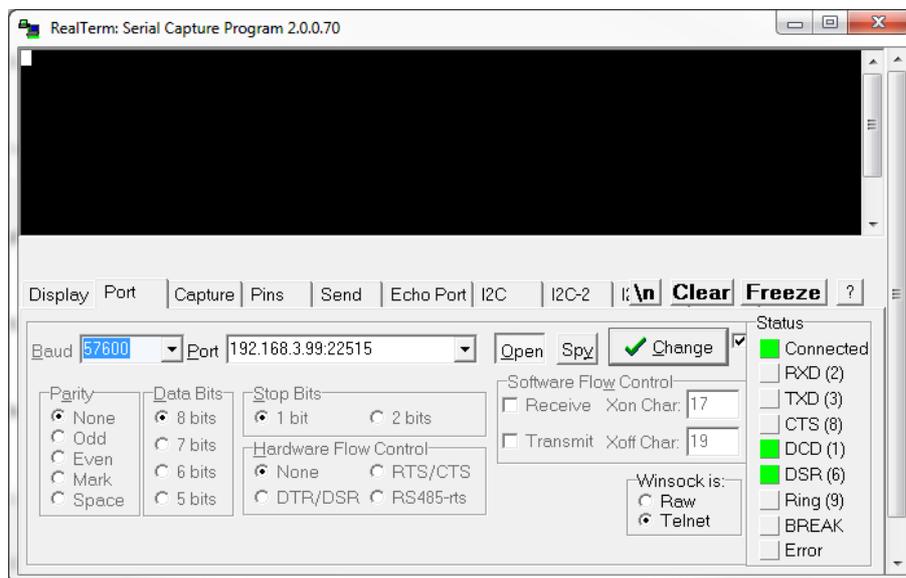
Dummy-Befehl	?Nop1{CRLF}	=Nop1#OK{CRLF}	Hilfreiche für Test der Kommunikation sowie als "Heartbeat".
Messwerte abfragen	?MVal1{CRLF}	=MVal1#0;-3;-12	Beispiel für 4 Messkanäle. Bei mehr Messkanäle ist die Antwort entsprechend länger.
Digitale Eingänge abfragen	?DIn1{CRLF}	=DIn1#0;1;0;1;1	Beispiel für 16 digitale Eingänge. Werte können 0 oder 1 sein.
Digitale Ausgänge setzen	?DOutSet1#1;1;0	=DOutSet1#1;1;0	In der Antwort wird der Ist-Zustand aller digitalen Ausgänge zurückgegeben. Dass können auch mehr sein, als in der Anfrage, wenn hier nur ein Teil der Ausgänge gesetzt wurde. Werte können 0 oder 1 sein.
Messwerte und digitale Eingänge gemeinsam abfragen	?MValDIn1{CRLF}	=MValDIn1#0;-3;	Kombination aus "MVal1" und "DIn1". Zuerst kommen die Messwerte, dann

			die digitalen Eingänge.
--	--	--	-------------------------



Tipp:

Zu Testzwecken kann der Datenaustausch auch via Telnet über zahlreiche Terminal-Programme erfolgen, wie zum Beispiel "RealTerm".



2.8.3 MscDLL Kurzübersicht



Dies ist lediglich eine Kurz-Übersicht!

Detaillierte Informationen finden Sie im Referenz-Handbuch der MscDLL.

Die MscDLL ist lange am Markt etabliert und wird von zahlreichen Anbietern von Mess-Software unterstützt.

Zur Kompatibilität mit diesen Software-Paketen unterstützt das Irinos EC diese Schnittstelle ebenfalls in eingeschränkter Form: es wird alles unterstützt außer die dynamische Messung. **D.h. das Irinos EC ist in Verbindung mit dieser Schnittstelle nur für statische Messungen geeignet.**

2.9 Diagnose und "Erste Hilfe"

Das Irinos-System hat zahlreiche interne Überwachungs-Mechanismen, die bei der Inbetriebnahme sowie im Service-Fall unterstützen können.

Ein dabei erkanntes [Diagnose-Ereignis](#)^[67] wird je nach Konfiguration im [Diagnose-Speicher](#)^[77] abgelegt und/oder über die [Software-Schnittstelle](#)^[60] zur PC-Software ausgegeben.

Die werksseitige Konfiguration der Diagnose-Ereignisse ist so ausgeführt, dass es für die typischen Anwendungsfälle keiner Um-Konfiguration bedarf.

2.9.1 Diagnose-Ereignisse

Jede Irinos-Box hat einen zentralen Ereignis-Handler. Sobald an einer Stelle in der Firmware ein besonderes Ereignis auftritt, wird dieses an den Ereignis-Handler gemeldet. Je nach Konfiguration wird das Ereignis

- dem Anwender bzw. der Applikation weitergemeldet und/oder
- in den [Diagnose-Speicher](#)^[77] eingetragen.

Im Normalbetrieb sollte kein Ereignis auftreten.

Um die Ereignisse auseinanderhalten zu können, gibt es verschiedenen Ereignis-Typen, die anhand der Ereignis-Nummer unterschieden werden.

Bei entsprechender Konfiguration wird das Auftreten eines Ereignisses über die [Status-LED](#)^[34] der Irinos-Box angezeigt. Das Ereignis kann auch über die [Software-Schnittstelle](#)^[60] ausgelesen werden.

Das Irinos-System hat eine sehr ausführliche Fehlerbehandlungs-Strategie um einen zuverlässigen Betrieb zu gewährleisten. Die meisten Ereignisse, die bei einem Fehler ausgelöst werden, sind hypothetischer Natur. Diese sind deshalb nicht dokumentiert. Wenden Sie sich an den Support, wenn so ein Ereignis auftritt.

Im Folgenden sind diejenigen Ereignisse aufgelistet, die für die Praxis relevant sind:

Ereignis 1: System / „System“	
Beschreibung	Allgemeines System-Ereignis
Typ	Information
Auslöser	<ul style="list-style-type: none"> ○ System wurde gestartet ○ Diagnose-Speicher^[77] wurde gelöscht
Weitermeldung an Anwender/Applikation	Nein, nicht aktivierbar
Eintragung in Diagnose-Speicher	Ja, nicht änderbar

Ereignis 4: MscDll – Kommunikation / „MscDll communication error“	
Beschreibung	Bei der Kommunikation zwischen dem Irinos-System und der MscDll wurde ein Problem festgestellt.
Typ	Fehler
Auslöser / Behebung	<ul style="list-style-type: none"> ○ Ein ungültiger Opcode wurde verwendet (Hilfstext im Diagnose-Speicher: „Invalid opcode in RX

	<p>packet“) -> Verwenden Sie nur gültige Opcodes</p> <ul style="list-style-type: none"> ○ Die Sende- oder Empfangs-Puffergröße ist zu klein (Hilfstext im Diagnose-Speicher: „Too much TX data“) -> Verwenden für die Konfigurationsdatei Msc.cfg die im Referenzhandbuch angegebene Port-Nummer und Puffer-Größen.
Weitermeldung an Anwender/Applikation	Ja, nicht de-aktivierbar
Eintragung in Diagnose-Speicher	Ja, änderbar

Ereignis 12: Box-Adressierung und -Terminierung / „EC-Link module detection error“	
Beschreibung	Beim Start des Irinos-Systems ist ein Problem bei der Erkennung oder Terminierung der Slave-Box(en) aufgetreten.
Typ	Fehler
Auslöser / Behebung	<ul style="list-style-type: none"> ○ Fehlerhafte EC-Link-Verkabelung oder defektes EC-Link-Kabel -> Überprüfen Sie die EC-Link-Verkabelung^[46] ○ Mehrere Master-Boxen in einem System -> Es ist nur eine Master-Box^[30] je Irinos-System zulässig

	<ul style="list-style-type: none"> ○ Irinos-Box defekt -> Tauschen Sie die Irinos-Box
Hinweis	<p>Bei Irinos-Systemen mit mehreren Slave-Boxen:</p> <p>Testen Sie das System zunächst mit 1, dann mit 2, dann mit 3, usw. Slave-Boxen um herauszufinden, an welcher Stelle das Problem auftritt.</p>
Weitermeldung an Anwender/Applikation	Ja, de-aktivierbar
Eintragung in Diagnose-Speicher	Ja, änderbar

Ereignis 13: EC-Link-Kommunikation / „EC-Link communication error“	
Beschreibung	Die Kommunikation über die EC-Link-Schnittstelle ist gestört.
Typ	Fehler
Auslöser / Behebung	<ul style="list-style-type: none"> ○ Fehlerhafte EC-Link-Verkabelung oder defektes ILink-Kabel -> Überprüfen Sie die EC-Link-Verkabelung^[46] ○ Unzureichende Spannungsversorgung (z.B. kurzzeitige Spannungseinbrüche) -> Verwenden Sie ein ausreichend dimensioniertes Netzteil^[31].
Hinweis	Die Link-Kommunikation hat eine integrierte Datenprüfung sowie eine Paket-Wiederholung im Fehlerfall.

	Wenn die Paket-Wiederholung mehrfach fehlschlägt, wird dieses Ereignis ausgelöst.
Weitermeldung an Anwender/Applikation	Ja, de-aktivierbar
Eintragung in Diagnose-Speicher	Ja, änderbar

Ereignis 15: Überlast Sinus-Oszillator / „Sine-oscillator“	
Beschreibung	Der Sinus-Oszillator für die induktiven Messtaster ^[37] wurde überlastet (Kurzschluss).
Typ	Fehler
Auslöser / Behebung	<ul style="list-style-type: none"> ○ Defekter Messtaster -> Tauschen Sie den defekten Messtaster. ○ Messtaster falsch angeschlossen, z.B. bei Verwendung einer Verlängerungs-Leitung -> Überprüfen Sie die Pin-Belegung
Hinweis	<p>Es wird zyklisch geprüft, ob der Oszillator-Kurzschluss noch vorhanden ist. Sobald er nicht mehr vorhanden ist, wird das Ereignis automatisch gelöscht.</p> <p>➔ Entfernen Sie zur Ursachensuche die Messtaster nacheinander. Warten Sie nachdem Entfernen eines Messtasters 10s. Sobald das Ereignis nicht mehr aktiv ist (erkennbar an der roten Fehler-LED ^[37]), wurde der defekte Messtaster entfernt.</p>

Weitermeldung an Anwender/Applikation	Ja, de-aktivierbar
Eintragung in Diagnose-Speicher	Ja, änderbar

Ereignis 24: Inkrementalgeber - Spannungsversorgung / „Inc. encoder power error“	
Beschreibung	Die Spannungsversorgung eines oder mehrerer Inkrementalgeber-Kanäle wurde aufgrund von Überlast / Kurzschluss abgeschaltet.
Typ	Fehler
Auslöser / Behebung	<ul style="list-style-type: none"> ○ Defekt eines Inkrementalgebers oder eines Inkrementalgeber-Kabels -> Defekten Inkrementalgeber tauschen. ○ Falscher Anschluss eines Inkrementalgebers. -> Anschlussbelegung prüfen. ○ Zu hohe Leistungsaufnahme der Inkrementalgeber -> Zulässige Anschlusswerte beachten.
Hinweise	<ul style="list-style-type: none"> ○ Bei einer Überlast / Kurzschluss an einem einzelnen Inkrementalgeber-Eingang wird nur der Eingang selbst deaktiviert. Alle anderen Eingänge sind weiterhin funktionsfähig. Sobald die Überlast bzw. der Kurzschluss beseitigt wurde, wird das Ereignis automatisch gelöscht.

	<ul style="list-style-type: none"> ○ Bei einer Gesamt-Überlast wird die Spannungsversorgung für die Inkrementalgeber-Eingänge dauerhaft abgeschaltet. Erst nach einem Neustart des Irinos-Systems sind die Inkrementalgeber-Eingänge wieder nutzbar.
Weitermeldung an Anwender/Applikation	Ja, de-aktivierbar
Eintragung in Diagnose-Speicher	Ja, änderbar

Ereignis 25: Inkrementalgeber – Signal(e) / „Inc. encoder application error“	
Beschreibung	Die Eingangssignale eines Inkrementalgeber-Eingangs waren / sind außerhalb des zulässigen Bereichs.
Typ	Fehler
Auslöser / Behebung	<ul style="list-style-type: none"> ○ Inkrementalgeber-Stecker wurde während des Betriebs abgezogen. ○ Inkrementalgeber-Steckverbinder sitzt lose (Wackelkontakt). -> Steckverbinder ordnungsgemäß verriegeln ○ Zu hohe Eingangsfrequenz⁸³⁾ des Inkrementalgebers -> Bewegung des Inkrementalgebers verlangsamen / Mechanischen „Schlag“ vermeiden ○ Zu lange Inkrementalgeber – Leitung -> Kurze Leitung verwenden

	<ul style="list-style-type: none"> ○ Falsche Anschlussbelegung des Inkrementalgebers -> Anschlussbelegung prüfen ○ Ausgangssignale des Inkrementalgebers außerhalb der Spezifikation -> Signale mit dem Irinos-Tool prüfen
Hinweise	<ul style="list-style-type: none"> ○ Die Inkrementalgeber-Signale können mit dem Irinos-Tool geprüft werden (nur 1Vss). ○ Beachten Sie die Applikationshinweise^[82] für Inkrementalgeber. ○ Ein Inkrementalgeber-Eingang kann über die Software-Schnittstelle^[60] zurückgesetzt werden (siehe jeweiliges Referenzhandbuch). Mit NmxDLL: NMX_ChannelSetParameter Mit MscDLL: Opcode opcSP
Weitermeldung an Anwender/Applikation	Ja, de-aktivierbar
Eintragung in Diagnose-Speicher	Ja, änderbar

Ereignis 27: Firmware-Update fehlgeschlagen / „Firmware update error“	
Beschreibung	Bei der Ausführung des Firmware-Updates ist ein Fehler aufgetreten.
Typ	Fehler

Auslöser / Behebung	<ul style="list-style-type: none"> ○ Falsche Firmware-Datei -> Verwenden Sie eine gültige Firmware-Datei ○ Übertragungsfehler -> Wiederholen Sie das Firmware-Update
Hinweise	Nach einem fehlgeschlagenen Firmware-Update ist weiterhin die „alte“ Firmware-Version aktiv.
Weitermeldung an Anwender/Applikation	Ja, nicht de-aktivierbar
Eintragung in Diagnose-Speicher	Ja, nicht änderbar

Ereignis 28: Firmware-Update erfolgreich / „Firmware update successful“	
Beschreibung	Ein Firmware-Update wurde erfolgreich durchgeführt.
Typ	Information
Weitermeldung an Anwender/Applikation	Nein, nicht aktivierbar
Eintragung in Diagnose-Speicher	Ja, nicht änderbar

Ereignis 36: Überlast der digitalen Ausgänge / „Digital I/O error“	
Beschreibung	Der Ausgangstreiber für die digitalen Ausgänge wurde überlastet

	(thermische Überlast).
Typ	Fehler
Auslöser / Behebung	<ul style="list-style-type: none"> ○ Zu hohe Dauerbelastung der digitalen Ausgänge. -> Maximale Ausgangslast an die Spezifikation anpassen.
Hinweise	Sobald der Ausgangstreiber abgekühlt ist, werden die Ausgänge automatisch wieder freigegeben und das Ereignis gelöscht.
Weitermeldung an Anwender/Applikation	Ja, de-aktivierbar
Eintragung in Diagnose-Speicher	Ja, änderbar

Ereignis 43: Sampling-Konfiguration / "NMX sampling configuration invalid"	
Beschreibung	Die Konfiguration der Echtzeit-Messwertaufnahme ist fehlerhaft.
Typ	Fehler
Auslöser / Behebung	<ul style="list-style-type: none"> ○ Ungültige Abtastrate
Hinweise	
Weitermeldung an Anwender/Applikation	Ja, de-aktivierbar

Eintragung in Diagnose-Speicher	Ja, änderbar
--	--------------

Ereignis 44: Sampling-Konfiguration / "NMX sampling error"	
Beschreibung	Fehler während der Ausführung der Echtzeit-Messwertaufnahme.
Typ	Fehler
Auslöser / Behebung	<ul style="list-style-type: none"> ○ Interner Puffer voll, da Messwerte nicht rechtzeitig abgeholt wurden. ○ Kommunikationsfehler zwischen mehreren Boxen, z.B. Kabel abgezogen/defekt.
Hinweise	
Weitermeldung an Anwender/Applikation	Ja, de-aktivierbar
Eintragung in Diagnose-Speicher	Ja, änderbar

2.9.2 Diagnose-Speicher

Jede Irinos-Box hat einen integrierten, nicht-flüchtigen Diagnose-Speicher, in welchen die aufgetretenen Ereignisse eingetragen werden (sofern das Speichern für das jeweilige Ereignis aktiviert ist). Er kann über den [Webserver](#)^[55] sowie über das [Irinos-Tool](#)^[54] ausgelesen werden.

Der Diagnose-Speicher ist damit ein wichtiges Hilfsmittel, um auftretende Probleme nachvollziehen und deren Ursache eingrenzen zu können. Dies gilt ins besonders dann, wenn ein Fehler sporadisch auftritt.

Je Irinos-Box können mindestens 32 Einträge im Diagnose-Speicher abgelegt werden. Sobald er voll ist, werden die ältesten Einträge automatisch gelöscht und damit Platz für neue Einträge geschaffen.

Ein Diagnose-Eintrag enthält neben dem eigentlichen Diagnose-Ereignis einige Zusatz-Informationen, die bei der Ursachensuche behilflich sein können. Dazu gehören die System-Zeit (EC-Link-Zeit^[29]) und die Absolut-Zeit.

Die EC-Link-Zeit entspricht der Zeit seit Start des Irinos-Systems in μs . Sie ist auf allen Irinos-Boxen eines Systems einheitlich.

Die Absolut-Zeit enthält das Datum sowie die Uhrzeit des Diagnose-Eintrags. Da das Irinos-System keine Echtzeit-Uhr enthält, ist die Absolut-Zeit beim Start des Systems immer 0. Sie sollte dann vom PC aus geschrieben werden. Dies geschieht über die [Software-Schnittstelle](#)^[60]. Jeder nachfolgende Diagnose-Eintrag wird dann mit der Absolut-Zeit versehen.

Nach dem Einschalten wird in jeder Irinos-Box der Diagnose-Eintrag „System (1)“ mit dem Hilfstext „System started“ eingetragen. Damit ist nachvollziehbar, ob ein Ereignis vor oder nach dem letzten Einschalten des Irinos-Systems aufgetreten ist.

2.9.3 Erste Hilfe "Netzwerkverbindung"

➔ Die in diesem Kapitel beschriebene Vorgehensweise erhebt keinen Anspruch auf Vollständigkeit. Sie deckt vielmehr die typische erforderliche Vorgehensweise ab. Weitere Informationen zur Netzwerk-Verbindung entnehmen Sie dem Benutzerhandbuch des Irinos-Tools.

Verbindungsprobleme zum Irinos-System liegen in der Regel an einer oder mehrerer der folgenden Ursachen:

- Die [Netzwerk-Verkabelung](#)^[46] ist fehlerhaft.
- Die Netzwerk-Konfiguration des PCs passt nicht zur Netzwerk-Konfiguration des Irinos-Systems.
- Die Kommunikations-Einstellungen für die [Software-Schnittstelle](#)^[60] sind falsch.

Netzwerk-Verkabelung prüfen

- a) Prüfen Sie, ob die Netzwerk-Schnittstelle des Irinos-Systems mit der Netzwerk-Schnittstelle des PCs verbunden ist.
Bei erfolgreicher elektrischer Verbindung leuchtet oder blinkt die [Ethernet-LED](#) ³⁴ an der Master-Box. Fahren Sie mit dem nächsten Schritt erst fort, wenn dies der Fall ist.

Netzwerk-Konfiguration prüfen

- b) Starten Sie das Irinos-Tool. Dieses zeigt beim Start alle im Netzwerk gefundenen Irinos-Systeme an. Das von Ihnen verwendete Irinos-System sollte hier aufgelistet sein. Sie erkennen dies anhand der aufgelisteten MAC-Adresse. Diese muss mit der MAC-Adresse auf dem Typenschild der Master-Box übereinstimmen.
In der Liste wird auch die IP-Konfiguration des Irinos-Systems angezeigt.
- c) Führen Sie den Verbindungsaufbau zum Irinos-System über das Irinos-Tool durch.

Fahren Sie mit dem nächsten Schritt fort, wenn der Verbindungsaufbau fehlschlägt.

Fahren Sie ansonsten mit Schritt f) fort.

- d) Öffnen Sie die Netzwerk-Konfiguration des Irinos-Systems durch einen Doppelklick auf die zugehörige Tabellenzeile im Irinos-Tool:



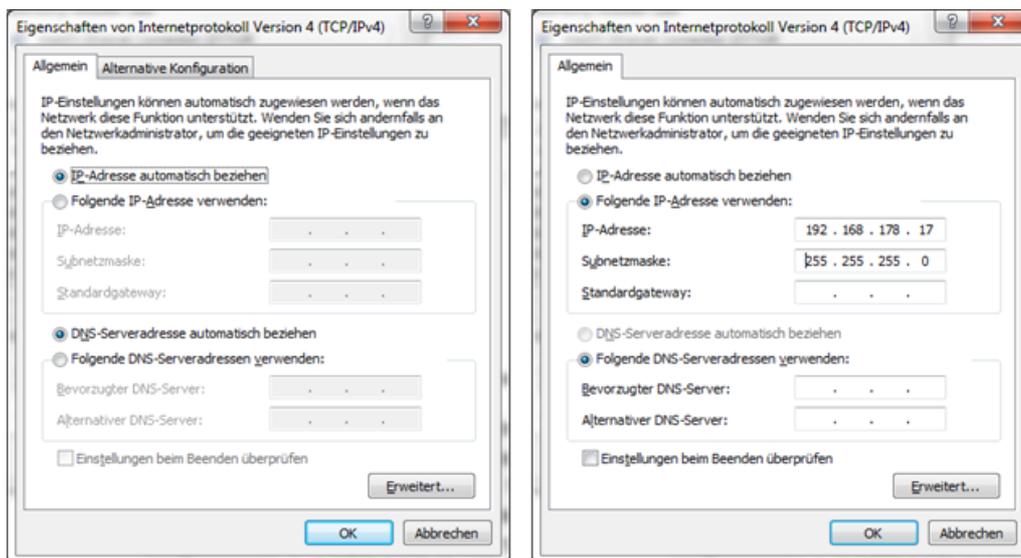
- e) Öffnen Sie die Windows Konfigurations-Einstellungen des Netzwerk-Adapters, an welchen das Irinos-System angeschlossen ist. Gehen Sie zu den Einstellungen zum „Internetprotokoll Version 4 (TCP/IPv4)“.

Wenn am Irinos-System der DHCP-Server konfiguriert ist, dann muss die Option „IP-Adresse automatisch beziehen“ ausgewählt sein (siehe

Abbildung links).

Wenn das Irinos-System eine feste IP-Adresse hat, dann muss auch am PC eine feste IP-Adresse hinterlegt werden. Beide IP-Adressen müssen im selben Subnetzbereich liegen. In den meisten Fällen wird die Subnetzmaske 255.255.255.0 verwendet. Dann müssen die ersten drei Stellen der IP-Adressen übereinstimmen. Hat das Irinos-System z.B. die IP-Adresse 192.168.178.1, dann muss der PC eine IP-Adresse aus dem Bereich 192.168.178.2 bis 192.168.178.254 haben (siehe Abbildung rechts).

Ändern Sie die IP-Einstellungen unter Windows erforderlichenfalls.



Kommunikations-Einstellungen übernehmen

- f) Prüfen Sie die bei der Mess-Software hinterlegte IP-Adresse. Diese muss mit der IP-Adresse des Irinos-Systems übereinstimmen (Auslieferungszustand 192.168.3.99).
- a) Starten Sie nun den Verbindungsaufbau über die Applikation. Dieser sollte nun erfolgreich sein.

2.10 Wartung, Pflege und Entsorgung

Wartung

Das Irinos-System ist für den wartungsfreien Dauerbetrieb ausgelegt.

Es wird empfohlen in regelmäßigen Abständen die Befestigung der außenliegenden Stecker zu prüfen, z.B. monatlich. Sie vermeiden dadurch einen Verschleiß der Steckverbinder. Zudem wird das mögliche Auftreten von Fehlerquellen vorbeugend verhindert.

Pflege

	<p>Vorsicht</p>
	<p>Unbeabsichtigte Reaktion beim Reinigen des Irinos-Messsystem</p> <p>Wenn das Irinos-Messsystem beim Reinigen eingeschaltet ist, können Bedienelemente unbeabsichtigt ausgelöst werden.</p> <p>Das Irinos-Messsystem oder damit verbundene Komponenten können unbeabsichtigt reagieren. Personenschaden oder Maschinenschaden kann die Folge sein.</p> <p>Schalten Sie das Gerät vor der Reinigung aus.</p>

Führen Sie bei intensiver Nutzung die in folgender Tabelle aufgelisteten Reinigungsarbeiten durch.

Bei besonders verschmutzter Umgebung kann eine häufigere Reinigung erforderlich sein. Im Gegenzug können die Reinigungsintervalle bei Gelegenheitsnutzung oder sauberer Umgebung verlängert werden.

Intervall	Reinigung
3 Monate	<p>Reinigung der Steckverbinder-Oberfläche von Öl und Staub.</p> <p>Verwenden Sie zur Reinigung ein Papiertuch, das mit Spülmittelwasser befeuchtet wurde.</p> <p>Schalten Sie das Irinos-System erst wieder ein, wenn die Steckverbinder komplett trocken sind.</p>
Monatlich	<p>Reinigen Sie das Gehäuse mit einem Papiertuch, das mit Spülmittelwasser befeuchtet wurde.</p> <p>Verwenden Sie ein kratzfreies Tuch.</p>

Entsorgung



Entsorgen Sie sowohl das Irinos-System als auch das Zubehör über die Elektronikschrott-Verwertung Ihres jeweiligen Landes. Entsorgen Sie es keinesfalls über den Hausmüll.

2.11 Applikationshinweise

2.11.1 Inkrementalgeber

Inkrementalgeber sind zuverlässige und präzise Messmittel, wenn bereits bei der Projektierung der Applikation potentiellen Problemen entgegen getreten wird. Beachten Sie deshalb die folgenden Applikationshinweise:

- [Referenzierung](#)^[82]
- [Zulässige Eingangsfrequenz](#)^[83]
- [Interpolation von 1Vss-Signalen](#)^[83]

2.11.1.1 Referenzierung bei Absolutmessung

Inkrementalgeber sind keine Absolut-Messmittel. Um absolute Messwerte zu erhalten ist nach dem Einschalten sowie nach einem Signalfehler immer eine Referenzierung erforderlich. Die Irinos-Box EC-INC bietet folgende Möglichkeiten zur Referenzierung:

- Referenzierung über Referenzmarke

Der Zählwert wird beim Überschreiten der Referenzmarke auf 0 gesetzt.

- Referenzierung per Software-Vorgabe:

Der Zählwert kann per Software jederzeit gesetzt werden. Es kann sowohl der Wert 0 als auch jeder beliebige andere Wert gesetzt werden.

In Verbindung mit der [NmxDLL](#)^[62] und [MscDll](#)^[67] können beide Aktionen ausgeführt werden:

- NmxDLL: NMX_ChannelSetParameter
- MscDLL: Opcode opcSP (0x35)

Bitte beachten Sie, dass die Irinos-Box EC-INC nur die technische Möglichkeit zur Referenzierung bieten kann. Die Vorgehensweise zur Referenzierung des Messwertes hängt vom jeweiligen Messablauf ab. Dies muss daher bereits in der Planungsphase berücksichtigt werden. Berücksichtigen Sie dabei insbesondere auch die Vorgehensweise nach dem Auftreten eines Inkrementalgeber-Fehlers.

2.11.1.2 Eingangsfrequenz

Die Eingangsfrequenz der Inkrementalsignale (TTL / RS422) bzw. Teilungsperioden (1 Vss) ist begrenzt. Details dazu entnehmen Sie den technischen Daten der betreffenden Irinos-Box.

Bei den meisten Messvorgängen liegt die theoretische Eingangsfrequenz deutlich unterhalb des Grenzwertes. In der Praxis kann es jedoch durch ruckartige Bewegungen dazu kommen, dass diese überschritten wird. Beispiele hierfür sind:

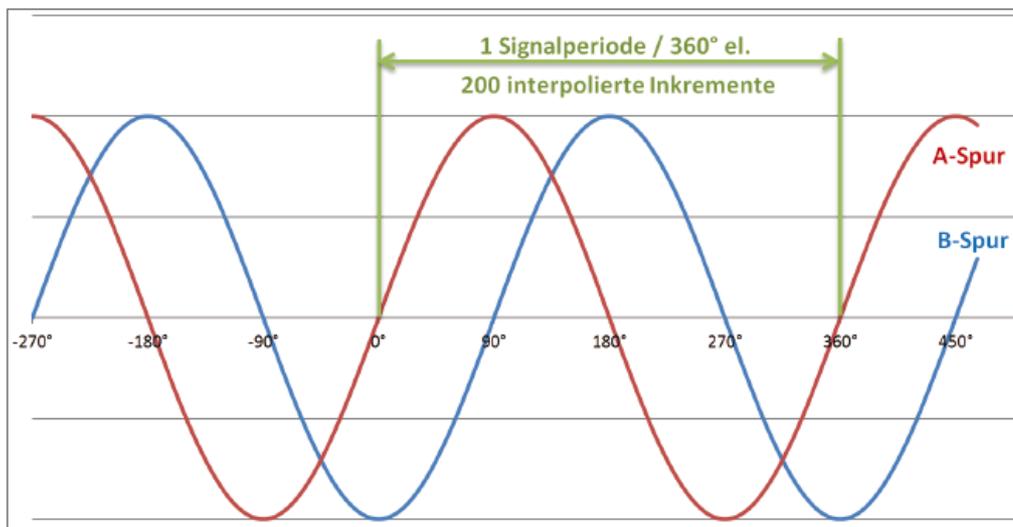
- "Losreißen" aus dem Stillstand (Überwindung der Haftreibung)
- Mechanischer Schlag
- Ruckartige Bewegung durch mechanische Spannungen

Wir empfehlen, dies bereits bei der Mechanik-Konstruktion zu berücksichtigen. Sollte eine ruckartige Bewegung nicht vermeidbar sein, so muss dies beim Messablauf bzw. bei der Messsignal-Auswertung berücksichtigt werden (z.B. durch Referenzierung während der Bewegung).

2.11.1.3 Interpolation (nur 1Vss)

Ein Inkrementalgeber mit 1 Vss - Schnittstelle gibt 2 sinusförmige Differenz-Signale aus, die um 90° zueinander phasenverschoben sind. Eine Signalperiode (d.h. 360°) entspricht dabei einer Inkrementalgeber-Teilung (siehe folgende Abbildung). Durch analoge Interpolation werden in der Irinos-Box EC-INC 200 Zwischenschritte innerhalb einer Signalperiode berechnet. Die nutzbare Auflösung eines Inkrementalgebers steigt damit um das 200-fache.

Beispiel: Ein Drehgeber hat eine angegebene Auflösung von 1.800 Teilungen/Umdrehung. Durch die Interpolation werden daraus $1.800 \text{ Teilungen/Umdrehung} * 200 \text{ Inkremente/Teilung} = 360.000$ Inkremente/Umdrehung.



Die Genauigkeit und Zuverlässigkeit der Interpolation hängt stark von den beiden analogen Differenz-Signalen ab. Ein ideales Signal zeichnet sich besonders durch folgende Eigenschaften aus:

- Der Differenzpegel jedes Signals beträgt 1 Vss.
- Der Signal-Offset ist 0, d.h. bei beispielsweise 0° hat das Signal immer denselben Wert.
- Die Phasenverschiebung der Signale ist immer exakt 90°.

Da ein derartiges Signal in der Praxis praktisch nie existiert, hat die Irinos-Box EC-INC eine patentierte interne Verstärkungs- und Offset-Regelung. Diese gleicht Abweichungen vom Ideal-Signal innerhalb der zulässigen Grenzwerte (siehe Datenblatt) aus.

Ausserhalb der Grenzwerte ist keine zuverlässige Interpolation möglich. Eine integrierte Signalüberwachung erkennt derartige Fehler. Diese können und sollten per Software im laufenden Betrieb ausgelesen werden. Im Fehlerfall sollte der Inkrementalgeber-Eingang zurückgesetzt und der Messtaster neu referenziert werden.

Signalqualität

Die Signalqualität hängt von vielen Faktoren ab. Dazu gehören insbesondere:

- Ist-Geschwindigkeit des Inkrementalgebers

Je höher die Ist-Geschwindigkeit des Inkrementalgebers, desto kleiner die tatsächliche Differenzspannung. Einige Inkrementalgeber haben im Stillstand und bei niedriger Geschwindigkeit eine gute Signalqualität. Sobald sie jedoch schnell bewegt / gedreht werden, verschlechtert sich die

Signalqualität deutlich.

- Mechanische Stabilität des Inkrementalgebers sowie der Mechanik

Ein unruhig laufender Drehgeber, ein unruhig laufender Messschlitten oder eine schlecht geführter Längenmesstaster führt zu Schwankungen im Messsignal.

- Einstellung bei offenen Gebersystemen

Bei offenen Gebersystemen (z.B. Glasmaßstäbe) muss der Sensor eingestellt werden. Eine ungenaue Einstellung kann vor allem bei dynamischen Betriebszuständen zu einem unzureichenden Sensorsignal führen.

- Kabellänge und Kabelqualität

Je länger das Kabel, desto schlechter die Signalqualität.

Je mehr Steckverbinder, desto schlechter die Signalqualität.

Ein unzureichend geschirmtes Kabel oder ein Kabel mit falscher Leitungsimpedanz verschlechtert das Messsignal.

Häufig sind auftretende Störungen auf eine Kombination der genannten Faktoren zurückzuführen.

Empfehlungen

- Beachten Sie die Grenzfrequenz des Inkrementalgebers. Diese entnehmen Sie den technischen Angaben des Herstellers.

Achtung: Die Grenzfrequenz ist abhängig von der Kabellänge.

- Kontrollieren Sie die Signalqualität bei der Inbetriebnahme. Die Signalpegel sollten bei der Inbetriebnahme noch ausreichend Abstand zu den Grenzwerten haben. Über das Irinos-Tool kann die Signalqualität mittels einer [Live-Anzeige](#)¹³⁵ bewertet werden. Details dazu entnehmen Sie dem Benutzerhandbuch des Irinos-Tools.
- Stellen Sie sicher, dass kein starker Ruck / Schlag auf den Inkrementalgeber einwirkt.

- Integrieren Sie in Ihren Messablauf bei Bedarf eine Möglichkeit zum "Fehler zurückzusetzen" und "Messwert referenzieren".
- Verwenden Sie kurze Leitungen mit ausreichender Schirmung (dies gilt auch für die Steckverbinder). Vermeiden Sie Kabelverlängerungen. Das Irinos-Konzept bietet die Möglichkeit, die Irinos-Box in räumlicher Nähe des Gebers unterzubringen.
- Halten Sie einen möglichst großen Abstand zwischen Inkrementalgeber-Leitung und potentiellen Störquellen, wie z.B. Umrichter und Motorleitungen.

2.11.2 Leistungsaufnahme

Die Leistungsaufnahme eines Irinos-Systems hängt von der Anzahl der angeschlossenen Irinos-Boxen und der Anzahl der angeschlossenen Verbraucher ab. Angeschlossene Verbraucher sind beispielsweise Messaufnehmer und Sensoren.

Eine Übersicht zur Abschätzung der Gesamt-Leistungsaufnahme entnehmen Sie folgender Tabelle. Bitte beachten Sie, dass es sich bei allen Werten um Richtwerte handelt. Die tatsächliche Leistungsaufnahme kann davon abweichen. Die genauen Angaben zur Leistungsaufnahme entnehmen Sie den jeweiligen Datenblättern.

Irinos-Box	Typische Leistungsaufnahme ohne angeschlossene Verbraucher	Empfohlener Kalkulationswert mit angeschlossenen Verbrauchern
EC-TFV ³⁷⁾	< 2 W	ca. 2 W

Es wird empfohlen im Rahmen der System-Inbetriebnahme den tatsächlichen Leistungsbedarf durch eine Messung zu überprüfen.

2.11.3 Speichern im nicht-flüchtigen Speicher

Die Anzahl an Schreibvorgängen in den nicht-flüchtigen Speichers des Irinos-Systems ist begrenzt. Das System ist so ausgelegt, dass der Grenzwert bei typischer Anwendung nie erreicht wird. Die maximale Anzahl an Schreibvorgängen ist in folgender Tabelle aufgelistet:

Systemfunktion	Maximale Anzahl an Schreib-Vorgängen	Bemerkung
Diagnose-Speicher	4,8 Millionen	
Messkanal-Konfiguration	200.000	Ausführung über <ul style="list-style-type: none">○ NmxDLL: NMX_ChannelSetConfig○ MscDLL: Opcode opcWCC
Netzwerk-Konfiguration	200.000	Ändern der IP-Adresse über das Irinos-Tool.
Firmware-Update	100.000	

2.12 Technische Daten

Die ausführlichen technischen Daten finden Sie im Datenblatt der jeweiligen Irinos-Box.

2.12.1 Allgemeine technische Daten

Messwertaufzeichnung	
Statische / Kontinuierliche Messung	Messrate ca. 30 Hz für flüssige Onlineanzeige
Echtzeit-Messung	<p>Bis 4.000 Messwerte/s auf allen Kanälen gleichzeitig, d.h.</p> <p>1 Kanal -> Gesamtmessrate 4.000 Messwerte/s</p> <p>17 Kanäle -> Gesamtmessrate 68.000 Messwerte/s</p> <p>32 Kanäle -> Gesamtmessrate 128.000 Messwerte/s</p> <p>Siehe dazu auch das Kapitel "Synchronisation und Geschwindigkeit²⁹".</p>
Synchronität	<p>Gleichzeitig Erfassung von allen Messkanälen</p> <p>Synchrone Messwernerfassung, auch über kaskadierte Irinos-Boxen.</p>

Kaskadierung / EC-Link-Schnittstelle	
Maximale Anzahl Irinos-Boxen	8
Maximale Anzahl Mess-Kanäle	Abhängig von der Kanalzahl der verwendeten Irinos-Boxen. Zum Beispiel bei EC-TFV maximal 64 Messkanäle.
Maximale Kabellänge EC-Link	10 m (Gesamtlänge der EC-Link - Linienverkabelung) Unter bestimmten Voraussetzungen auf Anfrage auch mehr.
Terminierung	Automatisch
Box-Adressierung	Automatisch
Gehäuse für EC-TFV³⁷, EC-INC	
Ausführung	Design-Gehäuse Aluminium schwarz eloxiert, Rückplatte rostfreies Blech, Frontplatte schwarz gepulvert
Abmessungen	120 x 85 x 49 mm (H x B x T)
Schutzart	In Anlehnung an IP54 Anschlüsse für Messtaster und digitale Ein-/Ausgänge entsprechend der jeweiligen Stecker-Spezifikation.
Befestigung Standard ⁴⁴	Über 2 Schrauben M4
Befestigung Zubehör ⁴⁴	Adapter für Hutschiene-Montage

Irinos-Tool Benutzerhandbuch

3 Irinos-Tool Benutzerhandbuch

3.1 Einleitung

3.1.1 Impressum

Titel	Irinos-Tool Benutzerhandbuch
Hersteller	Messtechnik Sachs GmbH Siechenfeldstraße 30/1 D-73614 Schorndorf Tel. 07181 / 99960-0 post@messtechnik-sachs.de
Gültig für	Irinos-Tool mit Messmodulen Irinos IR
Copyright-Hinweis	© 2019-2020 Messtechnik Sachs GmbH
Hinweis auf Markenzeichen und Warenzeichen	Alle in diesem Handbuch genannten Bezeichnungen von Erzeugnissen sind Warenzeichen der jeweiligen Firmen.
Material-Nr.	785-1003
Änderungshinweis	Technische Änderungen vorbehalten.
Stand der Drucklegung	05.06.2020

3.1.2 Revisions-Historie

Version	Datum	Änderungen
A	2016-02-17	Erste Version
B	2016-	<i>Zur Nutzung der folgenden Erweiterungen ist die IrinosTool-Version 2.0.1.7 oder neuer erforderlich.</i>

	04-06	<ul style="list-style-type: none">• Erweiterung des Irinos-Tools um die Anzeige/Ansteuerung von digitalen Ein-/Ausgängen^[129]• Erweiterung des Irinos-Tools um die Signalhistorie^[142] für 1Vss-Inkrementalgeber.
--	-------	--

3.1.3 Glossar

1Vss	1 Volt Spitze-Spitze
DHCP	Dynamical Host Configuration Protocol
DLL	Dynamic Link Library
DNS	Domain Name System
ILink	ILink-Schnittstelle zur Verbindung mehrerer Irinos-Boxen
IP	Internet Protocol
LAN	Local Area Network
MAC	Media Access Control
MDI	Medium Dependent Interface
MDI-X	Medium Dependent Interface Cross-Over (gekreuzt)
Auto-MDI(X)	Beschreibt die Fähigkeit, automatisch den Typ der Verbindungsleitung (gekreuzt oder ungekreuzt) zu erkennen und die Hardware entsprechend zu konfigurieren.
PC	Personal Computer
TCP	Transmission Control Protocol
TTL	Transistor-Transistor-Logik
UDP	User Datagram Protocol

3.1.4 Nutzungshinweise für Software / elektronische Dokumentation

Schutzrechte und Nutzungsumfang

Messtechnik Sachs stellt entweder auf portablen Datenträgern (z. B. Disketten, CD-ROMs, DVDs, ...), in schriftlicher (drucktechnischer) oder elektronischer Form Bedienungsanleitungen, Handbücher, Dokumentationen, sowie Softwareprogramme, alles und insgesamt im Folgenden als "LIZENZGEGENSTAND" bezeichnet, entgeltlich und/oder unentgeltlich zur Verfügung. Der LIZENZGEGENSTAND unterliegt u.a. urheberrechtlichen Schutzbestimmungen. Messtechnik Sachs oder Dritte haben Schutzrechte an diesem LIZENZGEGENSTAND. Soweit Dritten ganz oder teilweise Rechte an diesem LIZENZGEGENSTAND zustehen, hat Messtechnik Sachs entsprechende Nutzungsrechte. Messtechnik Sachs gestattet dem Verwender die Nutzung des LIZENZGEGENSTANDES unter den folgenden Voraussetzungen:

1.1) Nutzungsumfang elektronische Dokumentation

- a) Mit dem Erhalt/Erwerb oder der Überlassung eines LIZENZGEGENSTANDES erhalten Sie als Verwender in Bezug auf den jeweiligen LIZENZGEGENSTAND ein einfaches, nicht übertragbares Nutzungsrecht, das den Verwender berechtigt, diesen für eigene, ausschließlich betriebsinterne Zwecke, auf beliebig vielen Maschinen innerhalb seines Betriebsgeländes (Einsatzort), zu nutzen. Dieses Nutzungsrecht umfasst ausschließlich das Recht, den LIZENZGEGENSTAND auf den am Einsatzort eingesetzten Zentraleinheiten (Maschinen) zu speichern.
- b) Bedienungsanleitungen und/oder Dokumentationen, ungeachtet in welcher Form zur Verfügung gestellt, darf der Verwender an dessen Einsatzort außerdem in beliebiger Zahl über einen Drucker ausdrucken, sofern dieser Ausdruck vollständig mit diesen Nutzungsbedingungen und sonstigen Benutzerhinweisen ausgedruckt bzw. verwahrt wird.
- c) Mit Ausnahme des Messtechnik Sachs Logos ist der Verwender berechtigt, Bilder und Texte der Bedienungsanleitungen/Dokumentationen zur Erstellung eigener Maschinen- und Anlagendokumentation zu verwenden. Die Verwendung des Messtechnik Sachs Logos bedarf der schriftlichen Genehmigung von Messtechnik Sachs. Für die Übereinstimmung genutzter Bilder und Texte mit der Maschine/Anlage bzw. dem Produkt ist der Verwender selbst verantwortlich.
- d) Weitergehende Nutzungen sind in folgendem Rahmen zulässig:

Das Vervielfältigen ausschließlich zur Verwendung im Rahmen einer Maschinen- und Anlagendokumentation aus elektronischen Dokumenten sämtlicher dokumentierter Zulieferbestandteile. Die Demonstration gegenüber Dritten ausschließlich unter Sicherstellung, dass kein Datenmaterial ganz oder teilweise in anderen Netzwerken oder anderen Datenträgern verbleibt oder dort reproduziert werden kann.

Die Weitergabe von Ausdrucken an Dritte außerhalb der Regelung in Ziffer 3 sowie jede Bearbeitung oder andersartige Verwendung sind nicht zulässig.

1.2) Nutzungsumfang Softwareprodukte

An Software von Messtechnik Sachs jeglicher Art und der dazugehörigen Dokumentation erhält der Kunde ein nicht ausschließliches, nicht übertragbares und zeitlich nicht begrenztes Nutzungsrecht auf einem bestimmten bzw. im Einzelfall festzulegenden Hardware-Produkt. Messtechnik Sachs bleibt Inhaberin des Urheberrechts sowie aller anderen gewerblichen Schutzrechte. Das Recht Vervielfältigungen anzufertigen, ist nur zum Zwecke der Datensicherung gegeben. Copyright-Vermerke dürfen nicht entfernt werden.

2. Copyright Vermerk

Jeder LIZENZGEGENSTAND enthält einen Copyright Vermerk. Bei jeglicher Vervielfältigung die nach diesen Bestimmungen erlaubt ist, muss der entsprechende Copyright Vermerk des betreffenden Originals übernommen werden:

Bsp.: © 2015-2016, Messtechnik Sachs GmbH,

D-73614 Schorndorf

3. Übertragung der Nutzungsbefugnis

Der Verwender kann seine Nutzungsbefugnis nach diesen Bestimmungen bzgl. des jeweiligen LIZENZGEGENSTANDES in dem Umfang und mit den Beschränkungen der Bedingungen gemäß Ziffer 1 und 2 insgesamt auf einen Dritten übertragen. Auf diese Nutzungsbedingungen ist der Dritte ausdrücklich hinzuweisen.

II. Export LIZENZGEGENSTAND

Der Verwender muss beim Export des LIZENZGEGENSTANDES oder Teilen davon die Ausfuhrbestimmungen des ausführenden Landes und des Landes des Erwerbs beachten.

III. Gewährleistung

1. Produkte von Messtechnik Sachs werden hard- und softwaretechnisch weiterentwickelt. Liegt der LIZENZGEGENSTAND, gleich in welcher Form, einem Produkt nicht unmittelbar bei,

d.h. wird nicht auf einem, dem Produkt beiliegenden portablen Datenträger mit dem betreffenden Produkt als Liefereinheit ausgeliefert, gewährleistet Messtechnik Sachs nicht, dass eine elektronische Dokumentation mit jedem Hard- und Software-Stand des Produkts übereinstimmt.

2. Die in einer elektronischen Dokumentation enthaltenen Informationen können von Messtechnik Sachs ohne Vorankündigungen geändert werden und stellen keine Verpflichtung seitens Messtechnik Sachs dar.

3. Messtechnik Sachs gewährleistet, dass das von ihr erstellte Softwareprogramm mit der Anwendungsbeschreibung und Programmspezifikation übereinstimmt, jedoch nicht, dass die in der Software enthaltenen Funktionen vollständig unterbrechungs- u. fehlerfrei laufen oder dass die in der Software enthaltenen Funktionen in allen vom Erwerber gewählten Kombinationen und vorgesehenen Einsatzbedingungen ausführbar sind, bzw. den Erfordernissen entsprechen.

IV. Haftung/Haftungsbeschränkungen

1. Messtechnik Sachs stellt LIZENZGEGENSTÄNDE zur Verfügung, um den Verwender einerseits in die Lage zu versetzen Messtechnik Sachs Produkte die zum ordnungsgemäßen Betrieb einer Software bedürfen, diese vertragsgemäß einzusetzen, oder ihn bei der Erstellung seiner Maschinen- und Anlagendokumentation zu unterstützen. Für die elektronische Dokumentation, die in Form von portablen Datenträgern nicht unmittelbar einem Produkt beiliegt, d.h. nicht mit einem Produkt als Liefereinheit ausgeliefert wurde, gewährleistet

Messtechnik Sachs garantiert jedoch nicht, dass die separat vorgehaltene/gelieferte elektronische Dokumentation mit dem vom Verwender tatsächlich genutzten Produkt übereinstimmt.

Letzteres gilt insbesondere bei auszugsweisem Gebrauch für eigene Dokumentationen des Verwenders. Die Gewährleistung und Haftung für separat vorgehaltene / gelieferte portable Datenträger, d.h. mit Ausnahme der im Internet/Intranet vorgehaltenen elektronischen Dokumentation, beschränkt sich ausschließlich auf eine ordnungsgemäße Duplikation der Software, wobei Messtechnik Sachs gewährleistet, dass jeweils der neueste Stand der Dokumentation Inhalt des betreffenden, portablen Datenträgers ist. In Bezug auf die im Internet/Intranet vorgehaltene elektronische Dokumentation wird nicht gewährleistet, dass diese denselben Versionsstand aufweist wie die zuletzt drucktechnisch veröffentlichte Ausgabe.

2. Messtechnik Sachs haftet ferner nicht für mangelnden wirtschaftlichen Erfolg oder für Schäden oder Ansprüche Dritter wegen der Nutzung/Verwendung der vom Verwender eingesetzten LIZENZGEGENSTÄNDE, mit Ausnahme von Ansprüchen aus der

Verletzung von Schutzrechten Dritter, welche die Nutzung der LIZENZGEGENSTÄNDE betreffen.

3. Die Haftungsbeschränkungen nach Absatz 1. und 2. gelten nicht, soweit in Fällen von Vorsatz oder grober Fahrlässigkeit oder Fehlen zugesicherter Eigenschaften eine zwingende Haftung besteht. In einem solchen Fall ist die Haftung von Messtechnik Sachs auf den Schaden begrenzt, der für Messtechnik Sachs nach der Kenntnis der konkreten Umstände erkennbar war.

V. Sicherheitsrichtlinien/Dokumentation

Gewährleistungs- und Haftungsanspruch nach Maßgabe der vorstehenden Regelungen (Ziff. III. u. IV) sind nur gegeben, wenn der Anwender die Sicherheitsrichtlinien einer Dokumentation im Zusammenhang mit der Nutzung der Maschine und deren Sicherheitsrichtlinien oder die Nutzungsbedingungen von Software beachtet hat. Für die Kompatibilität nicht mit einem Produkt als Liefereinheit ausgelieferter elektronischer Dokumentation mit dem vom Anwender tatsächlich genutzten Produkt ist der Anwender selbst verantwortlich.

3.1.5 Vorwort

3.1.5.1 Zweck

	Warnung
	<p>Lesen Sie dieses Benutzerhandbuch sowie die zum Irinos-System zugehörige Dokumentation vor der Inbetriebnahme und Nutzung des Irinos-Systems vollständig durch. Dies gilt ins besonders für die darin enthaltenen Sicherheitshinweise.</p> <p>Fehlanwendung kann zu Schaden an Mensch, Maschine oder Anlage führen.</p>

3.1.5.2 Gültigkeitsbereich dieser Betriebsanleitung

Diese Bedienungsanleitung gilt für die Software "Irinos-Tool" in Verbindung mit dem industriellen Messsystem Irinos sowie zugehörigen Optionen. Details dazu entnehmen Sie der Original-Betriebsanleitung des Irinos-Systems.

3.1.5.3 Bestimmungsgemäßer Gebrauch

Irinos ist ein flexibles High-Speed - Messsystem für die industrielle Fertigungsmesstechnik. Es ist für den Dauerbetrieb (24/7) geeignet. Das Messgerät ist ausdrücklich nicht geeignet für den Einsatz in medizinischen oder explosionsgefährdeten Bereichen, Flugzeugen, für die

Raumfahrt sowie für den Heim- und Bürobereich. Nicht aufgeführte Bereiche, die diesen vom Sinn her ähnlich sind, gehören ebenfalls dazu.

In sicherheitskritischen Bereichen ist die Betriebssicherheit durch externe Vorrichtungen zu gewährleisten (z.B. externer Not-Aus-Kreis).

3.1.5.4 Erforderliche Grundkenntnisse

Für die Verwendung des Irinos-Tools sind allgemeine Grundkenntnisse in der Bedienung von Windows-Software erforderlich.

Für das Irinos-System gilt:

Für die elektrische Installation und die Inbetriebnahme sind Fachkenntnisse in Elektrotechnik sowie elektrotechnischer Sicherheit erforderlich.

Für die Einrichtung der Messaufgabe sind fundierte Kenntnisse in der industriellen Messtechnik sowie PC-Kenntnisse erforderlich.

3.1.5.5 Weitere Dokumentation

Beachten Sie das Begleitblatt, das mit jedem Irinos-Modul mitgeliefert wird. Dies gilt insbesondere für die darin enthaltenen Sicherheitshinweise. Die technischen Daten sind dem jeweils zugehörigen Datenblatt zu entnehmen.

Beachten Sie weiterhin die Original-Betriebsanleitung des Irinos-Systems.

3.1.5.6 Versionsstand

Diese Bedienungsanleitung bezieht sich auf den Firmware Versionsstand V1.0 und die Irinos-Tool - Version 2.0.

Screenshots können einen älteren Versionsstand zeigen. Diese werden nur aktualisiert, wenn sich die Benutzeroberfläche im Rahmen der Weiterentwicklung verändert.

3.2 Einführung

3.2.1 Über diese Hilfe

Diese Hilfe erläutert den Verbindungsaufbau zwischen dem Irinos-System und einem Windows-basierten PC, auf dem typischerweise die Messrechner-Software zum Einsatz kommt.

Weiterhin wird das Irinos Tool vorgestellt und detailliert beschrieben. Das Irinos Tool unterstützt den Anwender bei der Inbetriebnahme und bietet hierzu eine Reihe von Funktionen, wie

- Unterstützung beim Verbindungsaufbau und Verbindungstest,
- Visualisierung von Inventardaten,

- Änderungen der Gerätekonfiguration,
- Funktionstests,
- Firmware-Update.

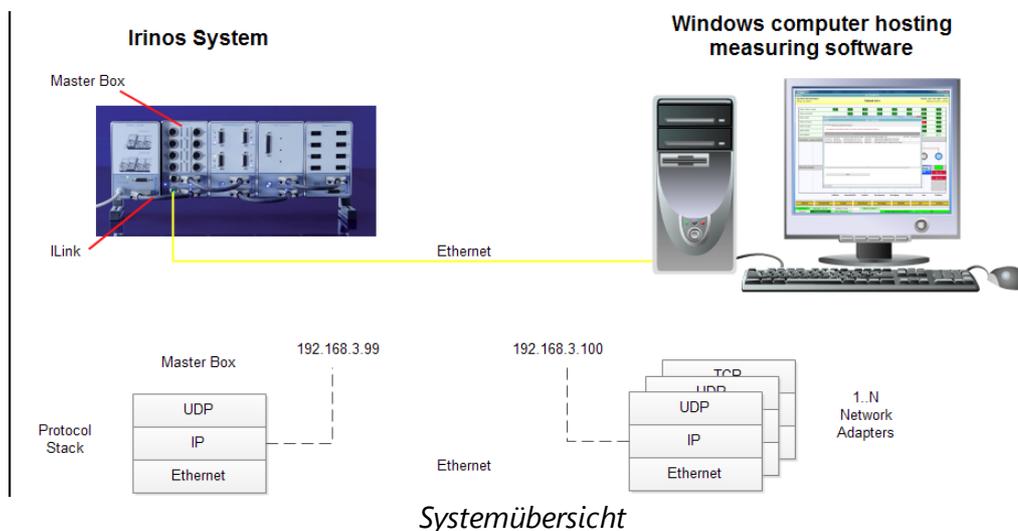
Die [Kurzanleitung](#)^[101] versetzt erfahrene Anwender in die Lage, das Irinos-System in wenigen Schritten in Betrieb zu nehmen und die Verbindung zum PC herzustellen.

Eine ausführliche Beschreibung zur [Netzwerkverbindung](#)^[106] und zum [Irinos-Tool](#)^[114] ergänzt die Kurzanleitung durch detailliertere Informationen.

Hinweis: Grundlegende Kenntnisse im Bereich IP-Netze, speziell der IP-Adressvergabe und dem Subnetz-Konzept sowie des DHCP-Protokolls sind Voraussetzung für das Verständnis der nachfolgenden Kapitel.

3.2.2 Übersicht

Wie in folgender Abbildung dargestellt, wird die physikalische Verbindung zwischen Irinos-System und PC über ein Ethernet-Kabel hergestellt, das auf der Irinos-Master-Seite mit einem M12-Stecker ausgestattet sein muss:



Als Kommunikationsprotokolle kommen UDP und IP zum Einsatz

Grundsätzlich müssen beide Kommunikationspartner konfiguriert werden, bevor eine Verbindung aufgebaut werden kann. Sowohl im Irinos-Master als auch im PC müssen zumindest IP-Adressen und Subnetzmasken gesetzt sein. Darüber hinaus besteht die Möglichkeit, ein Default Gateway festzulegen.

PC-seitig erfolgen diese Einstellungen über die Windows-Systemsteuerung an den jeweiligen Netzwerkadapters. Die entsprechenden Einstellungen in der Irinos-Master-Box werden mit dem Irinos-Tool durchgeführt. Neben der

Möglichkeit zur [Konfiguration der Netzwerkeinstellungen](#)^[115] bietet das Irinos-Tool [weitere Funktionen](#)^[114] zur Unterstützung der Inbetriebnahme.

Die manuelle Einstellung der Netzwerkkonfiguration stellt eine Möglichkeit dar, beide Kommunikationspartner mit den erforderlichen Kommunikationsparametern auszustatten. Nicht immer führt dies jedoch zu einem erfolgreichen Verbindungsaufbau. Fehlerhafte Netzwerkeinstellungen können dazu führen, dass sich das Irinos-System und der PC nicht im selben IP-Netz befinden, was einen Verbindungsaufbau unmöglich macht.

Abhilfe schafft hier das [DHCP](#)-Protokoll (Dynamic Host Configuration Protocol). DHCP ermöglicht eine automatisierte Erzeugung und Verteilung der Kommunikationsparameter. Innerhalb des DHCP-Modells gibt es einen DHCP-Server, der IP-Adressen aus einem Pool verwaltet und auf Anfrage an DHCP-Clients verteilt. Damit wird sichergestellt, dass IP-Adressen eindeutig sind und Server und Client sich im selben IP-Netz befinden.

Im obigen Szenario übernimmt das Irinos-System die Funktion des DHCP-Servers. Die Funktion ist bereits über die Werkseinstellungen aktiviert und kann direkt verwendet werden.

Lediglich auf der PC-Seite müssen die Einstellungen der Netzwerkkarte entsprechend konfiguriert werden, so dass sich der PC als DHCP-Client verhält und seine IP-Adresse automatisch bezieht.

3.3 Kurzanleitung

3.3.1 Voraussetzungen

Diese Kurzanleitung ist für Anwender vorgesehen, die sowohl mit dem Irinos-System als auch mit der IPv4-Netzwerkkonfiguration unter Microsoft Windows vertraut sind. Sie deckt Standard-Anwendungsfälle ab, die durch folgende Randbedingungen gekennzeichnet sind:

- Die Irinos Masterbox wird mit den werkseitigen Standardeinstellungen betrieben, d.h. der DHCP Server in der Masterbox ist aktiv.
- Die voreingestellte IP-Konfiguration, wie sie durch den DHCP-Server hergestellt wird
 - Irinos Box 192.168.3.99,
 - Netzwerkkarte des PC 192.168.3.100

deckt sich mit Anforderungen des Kundennetzes innerhalb dessen PC und Irinos Box betrieben werden sollen.

- Die Irinos Masterbox ist über eine geeignete Verbindungsleitung direkt mit einer Netzwerk-Schnittstelle des PC verbunden.

Hinweis: Ältere Netzwerkkarten ohne Auto-MDI(X) –Unterstützung erfordern ggf. eine gekreuzte Ethernet-Verbindungsleitung (sog. Cross-Over-Kabel). In der Praxis werden solche Netzwerkkarten heute kaum mehr eingesetzt.

- Der Anwender ist mit der prinzipiellen Funktion von DHCP vertraut und in der Lage DHCP an der entsprechenden Netzwerkschnittstelle zu aktivieren.

Für sämtliche Fragestellungen, die über diese Standardkonfiguration hinausgehen, wird die ausführliche Beschreibung in dieser Hilfe empfohlen, in welcher der Verbindungsaufbau detailliert dargestellt wird.

3.3.2 Einstellungen an der PC-Netzwerkkarte

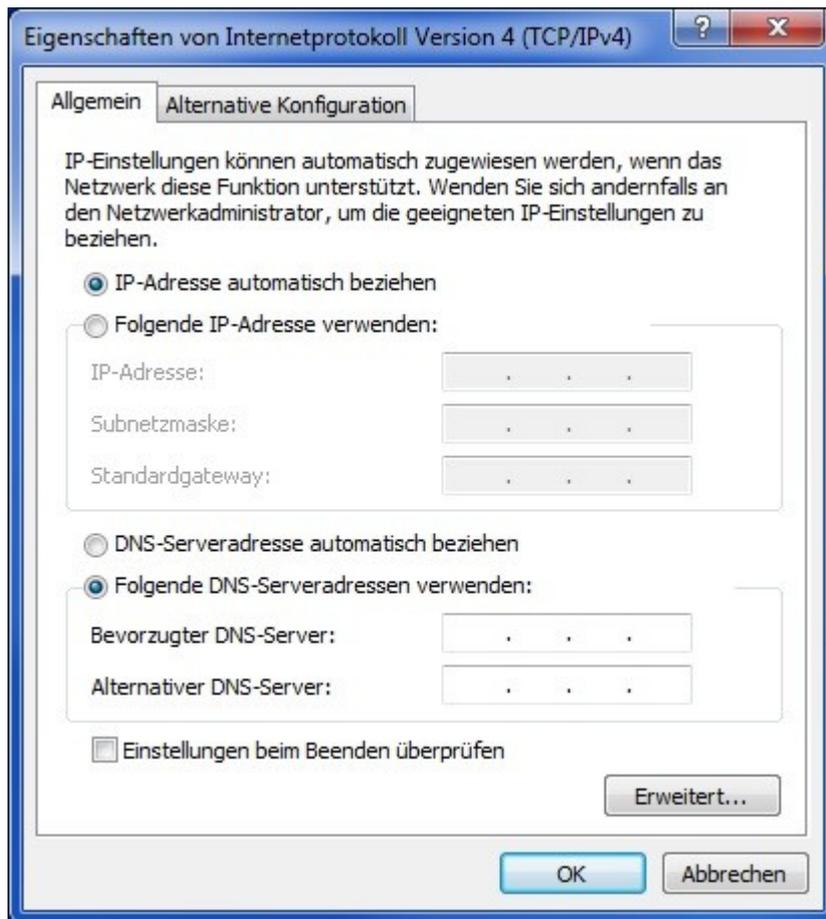
Um zu ermöglichen, dass ein Windows-PC seine Netzwerkkonfiguration von einem aktivierten DHCP-Server bezieht, muss dies über die Einstellungen der Netzwerkkarte entsprechend konfiguriert werden.

Abhängig von der Windows-Installation ist dies oft bereits der vordefinierte Wert. Eine Überprüfung dieser Einstellung ist jedoch empfehlenswert.

Zu diesen Einstellungen gelangt man wie folgt:

1. Systemsteuerung öffnen, Netzwerk- und Freigabecenter klicken dann „Adaptoreinstellungen ändern“
2. Entsprechenden Adapter (Netzwerkkarte) auswählen (Links-Klick) und dann über rechten Maus-Klick die „Eigenschaften“ öffnen
3. „Internetprotokoll Version 4 (TCP/IPv4)“ auswählen und über rechten Maus-Klick die „Eigenschaften“ öffnen.

Damit öffnet sich folgendes Fenster:



IPv4 - Konfiguration (DHCP aktiv)

In der Karteikarte "Allgemein"

4. "IP Adresse automatisch beziehen" auswählen

Damit wird der Netzwerk-Adapter als DHCP-Client konfiguriert, der nun seine Netzwerk-Konfiguration von der Irinos-Masterbox bezieht.

Sollte diese Einstellung bereits aktiviert sein, sind keine Eingaben erforderlich und das Fenster kann mit „OK“ geschlossen werden

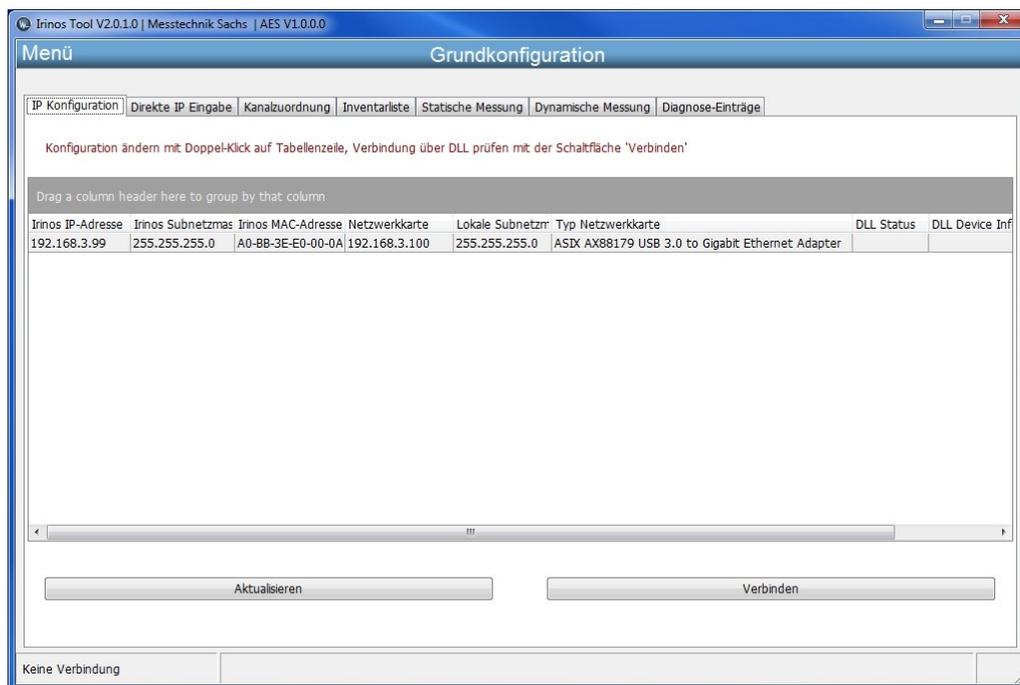
3.3.3 Irinos-Konfiguration und Verbindungsprüfung

Der Hauptvorteil des integrierten DHCP-Servers zeigt sich im vereinfachten Verbindungsaufbau zwischen Irinos-Masterbox und PC. Da die Einstellungen in der Irinos-Masterbox voreingestellt sind und damit der DHCP-Server aktiv ist, sind keine weiteren Schritte bei der Inbetriebnahme erforderlich.

Die [passende Konfiguration](#)^[102] der PC-seitigen Netzwerkkarten wird dabei vorausgesetzt.

Sobald die Masterbox an eine Spannungsversorgung angeschlossen und per Ethernet-Verbindungsleitung mit dem PC verbunden wurde, sollte es nun möglich sein, die Irinos-Masterbox mit dem Irinos-Tool anzusprechen. Dies erfolgt zunächst über eine Broadcast-Abfrage in alle angeschlossenen Netze.

Jede antwortende Masterbox wird als eigene Zeile im Startfenster des Irinos-Tools dargestellt (siehe folgende Abbildung). Prinzipiell ist es daher möglich, dass beliebig viele Masterboxen im IP-Netzwerk auf diese Abfrage antworten. In diesem Fall würde pro Box ein Eintrag im Startfenster dargestellt. In den meisten Anwendungen ist dies aber ein theoretischer Fall.



Startfenster des Irinos-Tool

Per Doppelklick auf diese Tabellenzeile kann nun die IP-Konfiguration der Irinos-Masterbox abgerufen werden. Folgende Abbildung zeigt das entsprechende Konfigurationsfenster. Es dient in erster Linie zur Überprüfung der Einstellungen. Solange der DHCP-Server aktiv ist (Auswahlfeld „DHCP“ steht auf „DHCP-Server“) sind keine Änderungen erforderlich und das Fenster kann wieder geschlossen werden.

Sollten Änderung an den Netzwerkeinstellungen der Masterbox erforderlich werden, könnten diese über dieses Fenster getätigt werden, nachdem das Auswahlfeld „DHCP“ auf „DHCP off“ gestellt wird.



DHCP	DHCP Server
Irinos IP-Adresse	192.168.3.99
Irinos Subnetzmaske	255.255.255.0
Default Gateway	0.0.0.0
Irinos MAC-Adresse	A0-BB-3E-E0-00-0A
Netzwerkkarte	192.168.3.100
Lokale Subnetzmaske	255.255.255.0

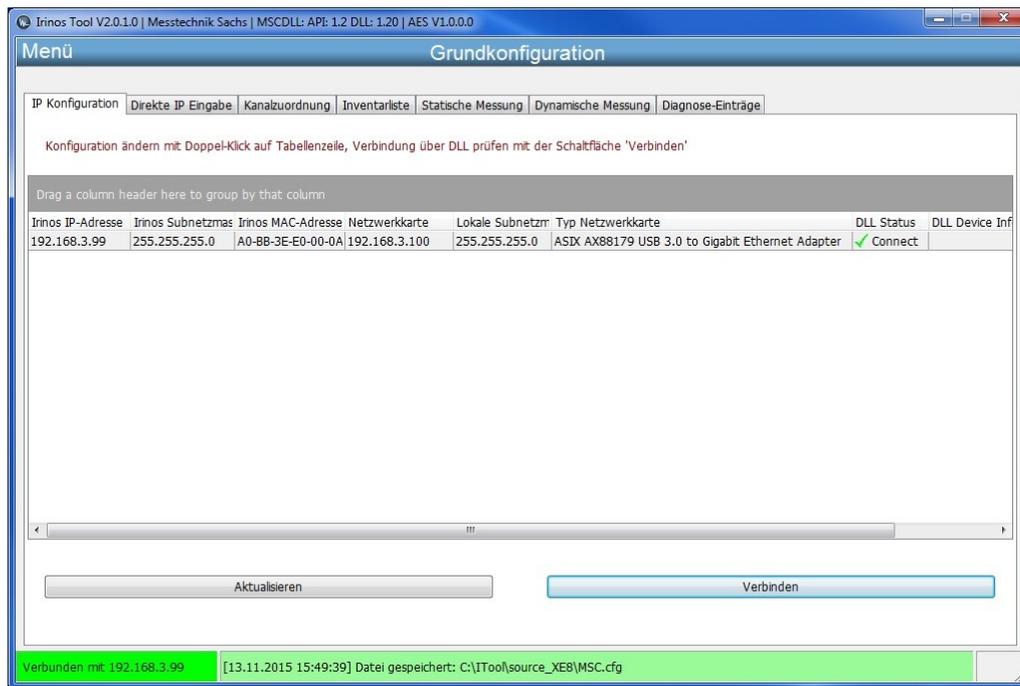
Schließen Senden

IP-Konfigurations - Fenster

Nachfolgend sollte eine Überprüfung der Verbindung über die MscDII erfolgen. Die MscDII ist Bestandteil der Irinos-Tool-Installation und wird als software-seitige Schnittstelle zur Irinos-Masterbox verwendet. Damit wird sichergestellt, dass eine beliebige Messrechner-Software in der Lage ist, auf die Irinos-Masterbox zuzugreifen.

Der Test der Verbindung wird über die Schaltfläche „Verbinden“ gestartet. Für den Fall, dass mehrere Irinos-Masterboxen vom Irinos-Tool zur Auswahl angeboten werden, muss die entsprechende Box zuerst per Mausklick selektiert werden.

Bei erfolgreicher Verbindung zeigt sich folgende Darstellung:



Verbindungstest über die MscDll erfolgreich

Das Ergebnis der Verbindungsprüfung wird in der Spalte „DLL Status“ angezeigt. Im Zuge der Verbindungsprüfung wird auch die Datei Msc.cfg erzeugt. Diese Datei dient als Konfigurationsdatei für die MscDll und enthält die aktuelle IP-Adresse der Masterbox.

Der Ablageort der Datei wird in der unteren Statuszeile des Irinos-Tools angezeigt. Vom dort angegebenen Pfad kann die Datei bei Bedarf kopiert und in den von der Messtecher-Software benötigten Pfad kopiert werden.

3.4 PC-seitige Netzwerkanbindung

3.4.1 Ethernet-Verbindung

Zur Verbindung von Irinos-Masterbox und PC kommt eine Ethernet-Verbindungsleitung nach Industrie-Standard zum Einsatz, die auf der Irinos-Master-Seite mit einem M12-Stecker (D-Codiert) ausgestattet sein muss.

Moderne Netzwerkkarten unterstützen durchgängig Auto-MDI(X). Damit entfällt die Notwendigkeit direkte Verbindungen zwischen PC und Irinos-Masterbox über Cross-Over-Leitungen zu führen. Lediglich bei Netzwerkkarten älterer Bauart ohne Auto-MDI(X)-Unterstützung kann dies erforderlich sein.

3.4.2 Netzwerkschnittstellen

Die Kommunikation über Netzwerk setzt Netzwerkeinstellungen voraus, ohne die keine Kommunikation möglich ist. Im Wesentlichen erfordert das verwendete IP-Protokoll die Festlegung von IP-Adresse, Subnetzmaske und Default Gateway.

Kommen mehrere Netzwerkkarten (im Folgenden auch als Netzwerkadapter bezeichnet) zum Einsatz, gilt dies für jede Netzwerkkarte.

Die Netzwerkeinstellungen können in der Windows Systemsteuerung für jede Netzwerkkarte angepasst werden. Über das Symbol „Netzwerk und Freigabecenter“ und weiter über den Menüpunkt „Adaptoreinstellungen ändern“ gelangt man zur Darstellung der einzelnen Netzwerkkarten/-adaptern:



Alle Netzwerkverbindungen ohne Verbindung

Für jeden Netzwerkadapter wird ein entsprechendes Symbol angezeigt. Ein rotes Kreuz weist darauf hin, dass keine Ethernet-Verbindung zu einem anderen netzwerkfähigen Gerät besteht. Wird eine Ethernet-Leitung gesteckt, die mit einem anderen netzwerkfähigen Gerät verbunden ist, so verschwindet das rote Kreuz.

Zuordnung der Netzwerkanschlüsse ermitteln

Im Folgenden wird anhand eines Beispiels gezeigt, wie durch Stecken oder Entfernen des Netzwerkkabels die Zuordnung zwischen LAN-Anschluss und dem entsprechenden Symbol in der Systemsteuerung ermittelt werden kann:

1. Kein Netzwerkkabel gesteckt

Alle Netzwerkverbindungen sind mit einem roten Kreuz versehen:



2. Verwendung der ersten Netzwerk-Schnittstelle

Das Netzwerkkabel ist in LAN1 eingesteckt. Deshalb ist "LAN-Verbindung 1" ohne rotes Kreuz dargestellt:



3. Verwendung der zweiten Netzwerk-Schnittstelle

Das Netzwerkkabel ist in LAN2 eingesteckt: Deshalb ist "LAN-Verbindung 2" ohne rotes Kreuz dargestellt:



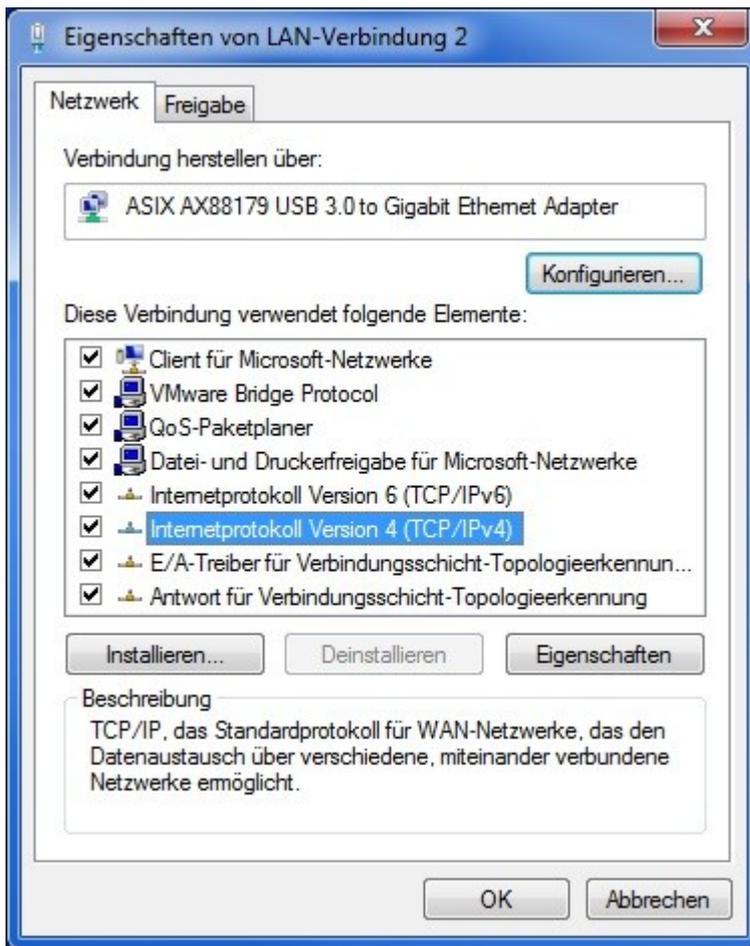
3.4.3 Netzwerkeinstellungen

3.4.3.1 IP-Konfiguration mit DHCP

Das Irinos-System unterstützt durch seinen integrierten DHCP-Server den einfachen und sicheren Verbindungsaufbau zwischen Irinos-System und PC. Der DHCP-Server ist werkseitig bereits aktiviert und liefert die Netzwerkeinstellungen für den angeschlossenen, PC-seitigen Netzwerkadapter.

Falls DHCP nicht verwendet werden soll, kann es über das Irinos-Tool [deaktiviert](#)^[115] werden. Die IP-Konfiguration muss dann [manuell](#)^[112] erfolgen.

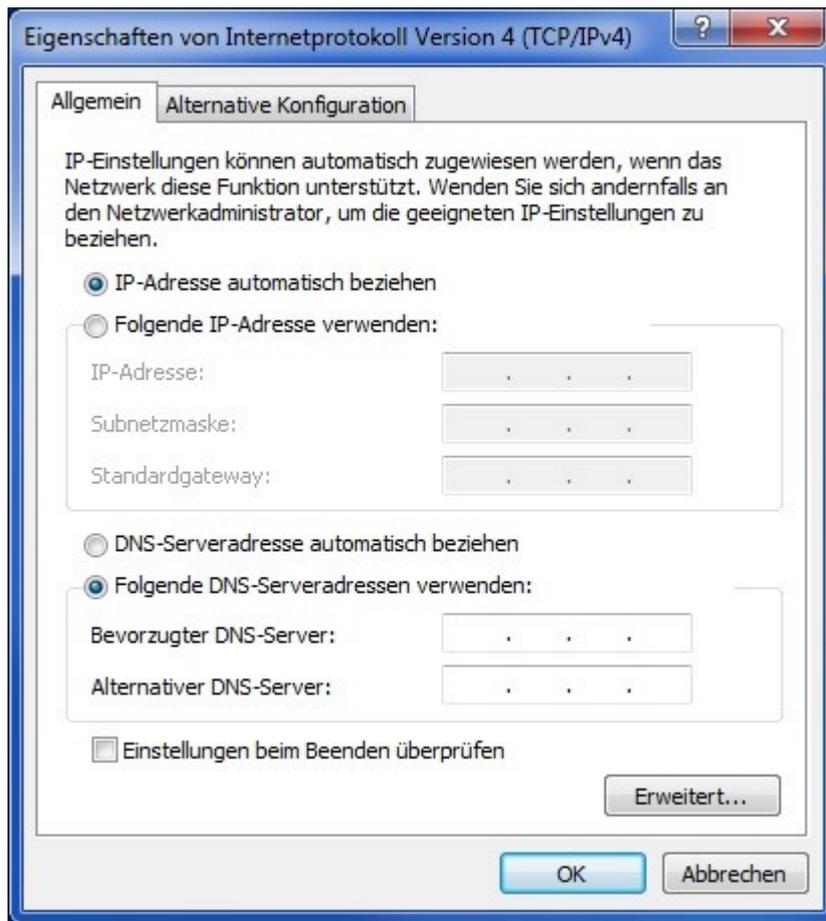
Wird DHCP verwendet, muss der entsprechende Netzwerkadapter als DHCP-Client konfiguriert werden. Ein Rechts-Klick auf das Symbol „LAN-Verbindung x“ und die Auswahl des Menüpunkts „Eigenschaften“ öffnet das nächste Konfigurationsfenster:



In diesem Fenster folgende Aktion ausführen:

-> **“Internetprotokoll Version 4 (TCP/IP)” auswählen und die Schaltfläche „Eigenschaften“ drücken.**

Dies führt schließlich zur folgenden Eingabemaske:



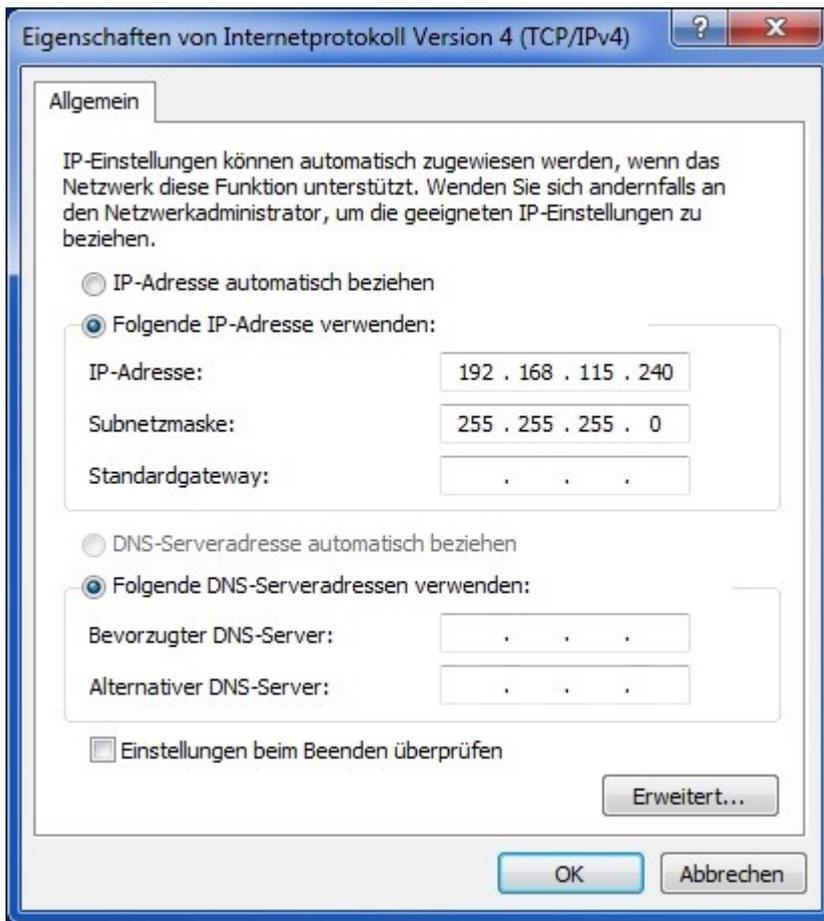
In der Karteikarte „Allgemein“

-> **“IP-Adresse automatisch beziehen“ auswählen.**

Damit ist der Netzwerkadapter als DHCP-Client konfiguriert.

3.4.3.2 IP-Konfiguration ohne DHCP

Falls der DHCP-Server des Irinos-Systems nicht verwendet werden soll, kann er über das Irinos-Tool [deaktiviert](#)^[115] werden. In diesem Fall müssen die Netzwerkeinstellungen „manuell“ konfiguriert werden. Ein Beispiel zeigt folgende Abbildung:



--> **“Folgende IP-Adresse verwenden:” auswählen**

Dann Werte für IP-Adresse und Subnetzmaske eingeben

Als Subnetzmaske wird generell der Wert 255.255.255.0 empfohlen.

Wichtig: Die IP-Adresse des Netzwerkadapters muss im gleichen Subnetz liegen wie die IP-Adresse des Irinos-Systems.

Wird die Subnetzmaske wie im obigen Beispiel gewählt, so ergibt sich als möglicher Bereich für die IP-Adresse

192.168.115.1

...

192.168.115.254.

Die IP-Adresse des Netzwerkadapters muss sich jedoch von der des Irinos-Systems unterscheiden.

Die Einträge für das Default Gateway und DNS-Server müssen nicht mit Werten belegt werden.

Zum Speichern der Einstellungen das Fenster mit “OK” beenden.

3.5 Irinos-Tool

3.5.1 Allgemeines

Das Irinos-Tool stellt dem Anwender eine Reihe von Funktionen zur Verfügung, die ihn bei der Inbetriebnahme unterstützen. Hierzu zählen die [Identifikation des Irinos-Systems im IP-Netz](#)^[114], die [Konfiguration der IP-Netzwerkeinstellungen](#)^[115] sowie [Verbindungstests](#)^[119] über die MscDII.

Darüber hinaus ermöglicht es die [Abfrage der Inventardaten](#)^[123], bietet Funktionen zur [Kanalkonfiguration](#)^[121] und unterstützt [einfache Funktionstests](#)^[127] sowie [Firmware-Updates](#)^[132].

Zugleich dient das Irinos-Tool dazu, Daten aus dem Irinos-System auszulesen und als [Konfigurationsdatei](#)^[119] der eigentlichen Messrechner-Software zur Verfügung zu stellen.

3.5.2 Installation

Das Irinos-Tool wird als komprimiertes, selbst-entpackendes Archiv geliefert. Die Installation erfolgt über einen Doppelklick auf die Archivdatei. Bei der Abfrage nach dem Zielordner muss ein **schreibbarer Ordner** angegeben werden, in den das Irinos Tool installiert werden kann. Unterhalb dieses Ordners werden bei Bedarf auch Firmware-Dateien abgelegt. Der ausgewählte Ordner wird in der weiteren Beschreibung als „MeinOrdner“ bezeichnet.

Es wird empfohlen, eine Verknüpfung zu der Datei „ITool.exe“ im Verzeichnis „MeinOrdner/ITool/source_XE8“ anzulegen und diese Verknüpfung auf dem Desktop abzulegen.

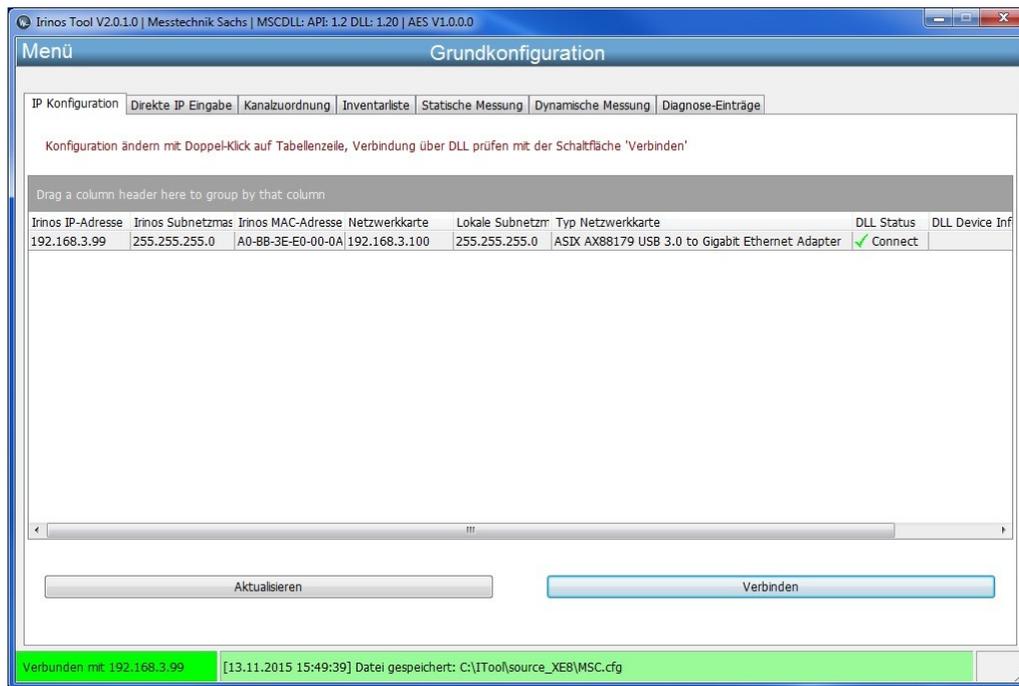
Beim ersten Starten des Irinos-Tools erscheint üblicherweise eine Windows-Firewall-Meldung. Hier die Schaltfläche „Zugriff zulassen“ drücken.

3.5.3 Starten des Irinos-Tools

Das Irinos-Tool kann entweder durch einen Doppelklick auf das [Desktop-Symbol](#)^[114] oder direkt auf die Datei MeinOrdner /ITool/source_XE8/ITool.exe gestartet werden.

Während des Startvorgangs werden über die aktiven Netzwerkadapter Broadcast-Abfragen in die angeschlossenen Netzwerke gesendet. Irinos-Boxen antworten auf diese Abfrage mit einer entsprechenden Antwort-Nachricht.

Aus dieser Antwort erzeugt das Irinos-Tool einen Eintrag in der Tabelle des Startfensters:



Startfenster des Irinos-Tools

Dabei repräsentiert jede Tabellenzeile eine Irinos-Box, von der folgende Informationen dargestellt werden.

- Irinos IP-Adresse
- Irinos Subnetzmaske
- Irinos MAC-Adresse
- IP-Adresse des Netzwerkadapters, über den die Irinos-Box verbunden ist.
- Subnetzmaske des Netzwerkadapters
- Typinformation des Netzwerkadapters

Mit der Schaltfläche "Aktualisieren" werden Änderungen in diesem Kontext abgerufen und angezeigt.

3.5.4 IP-Konfiguration

Da eine Irinos-Masterbox mit aktivierter DHCP-Funktion ausgeliefert wird, werden in der Regel keine Einstellungen an der IP-Konfiguration nötig sein.

Werden Änderungen an der IP-Konfiguration erforderlich, so öffnet ein Doppel-Klick auf die Zeile des Startfensters das Fenster der IP-Konfiguration. Dieses ist in folgender Abbildung dargestellt und zeigt den Initialzustand des Irinos-Systems: "DHCP aktiviert".

--> **Auswahlbox DHCP: DHCP Server**

The screenshot shows a window titled "IP-Konfiguration" with a blue header bar. Inside the window, there is a dropdown menu labeled "DHCP" with "DHCP Server" selected. Below this are several input fields:

- Irinos IP-Adresse: 192.168.3.99
- Irinos Subnetzmaske: 255.255.255.0
- Default Gateway: 0.0.0.0
- Irinos MAC-Adresse: A0-BB-3E-E0-00-0A
- Netzwerkkarte: 192.168.3.100
- Lokale Subnetzmaske: 255.255.255.0

At the bottom of the window, there are two buttons: "Schließen" (Close) on the left and "Senden" (Send) on the right.

Wird der DHCP-Server deaktiviert, in dem mit der DHCP-Auswahlbox der Wert "DHCP off" angewählt wird, so werden die Eingabefelder für

- die Irinos IP-Adresse
- die Irinos Subnetzmaske und
- das Default Gateway

freigeschaltet:

DHCP	DHCP off
Irinos IP-Adresse	192.168.3.99
Irinos Subnetzmaske	255.255.255.0
Default Gateway	0.0.0.0
Irinos MAC-Adresse	A0-BB-3E-E0-00-0A
Netzwerkkarte	192.168.3.100
Lokale Subnetzmaske	255.255.255.0

Schließen Senden

Die Irinos MAC-Adresse sowie die [IP-Adresse und Subnetzmaske des Netzwerkadapters](#) werden nur angezeigt und können nicht verändert werden. Sie dienen hier als Orientierung für die Auswahl einer geeigneten IP-Adresse des Irinos-Systems.

IP-Adresse, Subnetzmaske und Default Gateway können nun entsprechend den Applikations-Anforderungen eingegeben werden.

Bevor die Eingaben zum Irinos-System gesendet werden, führt das Irinos-Tool eine Reihe von Konsistenzprüfungen durch, die sich an folgenden Regeln orientieren:

- Nicht zulässige IP-Adressen wie 127.0.0.1, 127.0.0.0, 255.255.255.255 werden abgelehnt.
- Befindet sich die gewählte IP-Adresse im Subnetz des lokalen Netzwerkadapters?
- Ist die gewählte IP-Adresse verschieden von der des lokalen Netzwerkadapters?
- Liegt die gewählte IP-Adresse in einem Standard-IP-Bereich (10.0.0.0 bis 10.255.255.255, 172.16.0.0 bis 172.31.255.255, 192.168.0.0 bis 192.168.255.255)?

Dabei werden nicht zulässige IP-Adressen generell abgelehnt, alle anderen Einstellungen werden verwendet, sofern der Anwender dies explizit bestätigt.

Nachdem die IP-Konfigurationsdaten zum Irinos-System gesendet wurden, führt dieses einen Neustart durch, um die neuen Einstellungen zu übernehmen. Wird während dieser Zeit die Schaltfläche „Aktualisieren“ gedrückt, wird keine Irinos-Box gefunden und die Tabelle des Startfensters bleibt leer. Nach etwa 10 Sekunden ist die Irinos-Box wieder erreichbar. Nach erneutem Drücken der Schaltfläche „Aktualisieren“ werden die aktuellen Werte der IP-Konfiguration angezeigt.

3.5.5 Direkte IP-Eingabe

Die Funktion zur direkten IP-Eingabe stellt eine **Sonderfunktion dar, die für den normalen Betrieb nicht benötigt wird**. Sie kann dazu verwendet werden, die Werte für die IP-Konfiguration des Irinos-Systems direkt einzugeben, ohne die Auswahlfunktion des Startfensters zu verwenden.

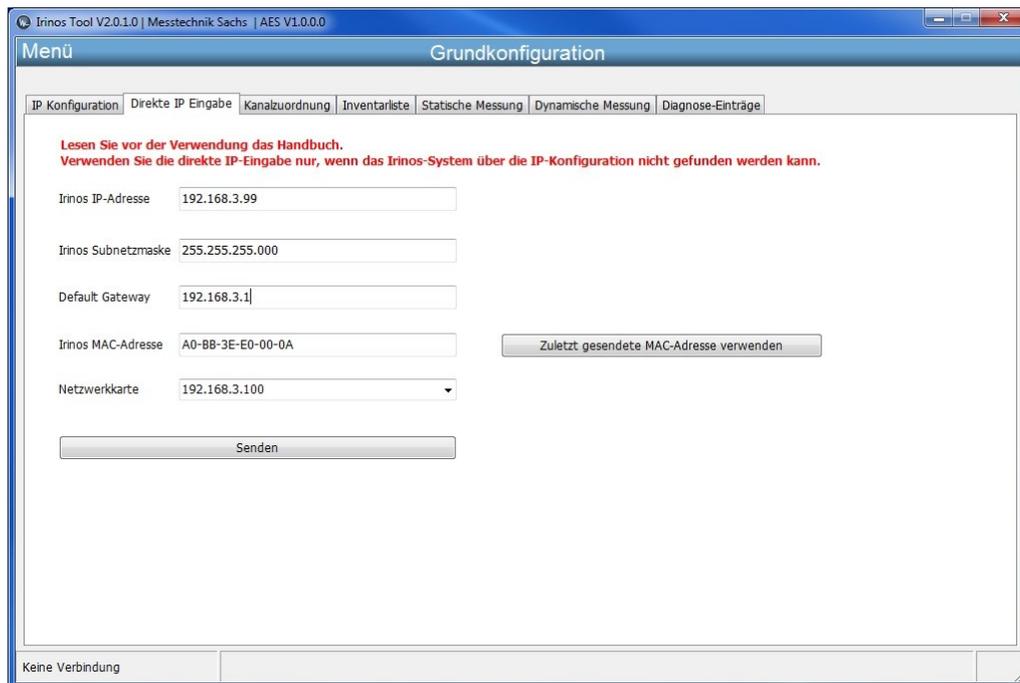
Die Eingabewerte müssen dabei vollständig und korrekt im Sinne der Adressvergaberegeln für IP-Netze durch den Anwender vorgegeben werden, eine Prüfung durch das Tool erfolgt nicht. Weiterhin ist bei mehreren Netzwerkkarte die Auswahl erforderlich, über welchen Netzwerkadapter die Nachricht gesendet werden soll.

Diese Funktion ist für Anwendungsfälle vorgesehen, bei denen ein Irinos-System über das Startfenster nicht auffindbar ist oder sich seine Konfiguration nicht ändern lässt, obwohl es mit einem Netzwerkadapter verbunden ist. Derartige Fälle können auftreten, wenn beispielsweise ein Irinos-System mit einer IP-Adresse konfiguriert wurde, die nicht im Subnetz des zugehörigen Netzwerkadapters liegt. Die direkte IP-Eingabe dient dazu, solche Fehlkonfigurationen zu beheben.

Hierzu werden die eingegebenen Werte per Broadcast-Nachricht über den ausgewählten Netzwerkadapter an das angeschlossene Netz gesendet. Über die Schaltfläche „Zuletzt verwendet MAC-Adresse verwenden“ kann die MAC-Adresse wiederhergestellt werden, an die zuletzt eine IP-Konfiguration gesendet wurde.

Die Irinos-Box empfängt die Broadcast-Nachricht und vergleicht dabei die in der Nachricht enthaltene MAC-Adresse mit der eigenen. Bei Übereinstimmung übernimmt die Irinos-Box die empfangenen Werte und führt einen Reset durch.

Wurden für die IP-Konfiguration richtige Werte verwendet, so sollte die Box nach diesem Reset wieder im Startfenster erscheinen. Hierzu im Startfenster die Schaltfläche „Aktualisieren“ drücken.



3.5.6 Verbindungsprüfung über DLL

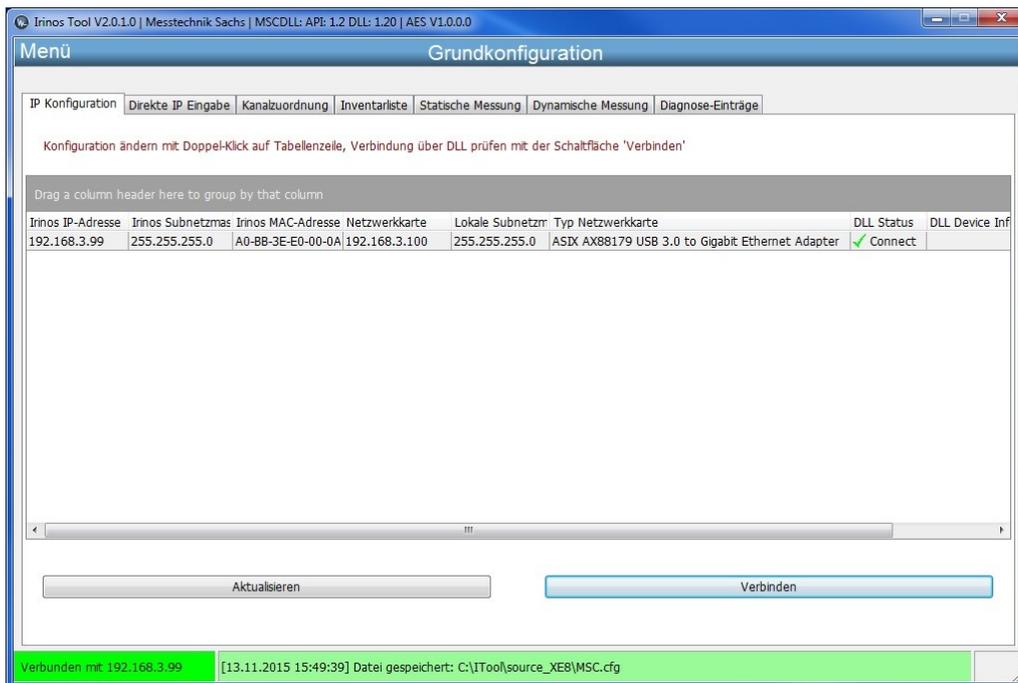
Als Bindeglied zwischen dem Irinos-System und einer beliebigen Messrechner-Software wird eine sogenannte „dynamisch gelinkte Library“ (kurz DLL) mit der Bezeichnung Mscdll.dll bereitgestellt.

Diese DLL enthält die grundlegenden Zugriffsmethoden zum Irinos-System, zum Verbindungsaufbau und zur Bereitstellung von Messkanälen zwischen Messsystem und Messrechner-Software. Aus diesem Grund ist es sinnvoll, bei der Inbetriebnahme den Verbindungsaufbau über die DLL zu testen, bevor die eigentliche Messrechner-Software auf die Irinos-Box zugreift.

Als Voraussetzung für die Kommunikation über die DLL muss die IP-Adresse des Irinos-Systems der DLL bekannt gemacht werden. Diese erfolgt über eine Konfigurationsdatei „Msc.cfg“, in der die IP-Adresse abgelegt wird. Aus dieser Datei liest die DLL während der Startphase die IP-Adresse des Irinos-Systems und baut die Verbindung eigenständig auf.

Das Irinos-Tool übernimmt während der Verbindungsprüfung das Abspeichern der IP-Adresse des (ausgewählten) Irinos-Systems in die Datei Msc.cfg.

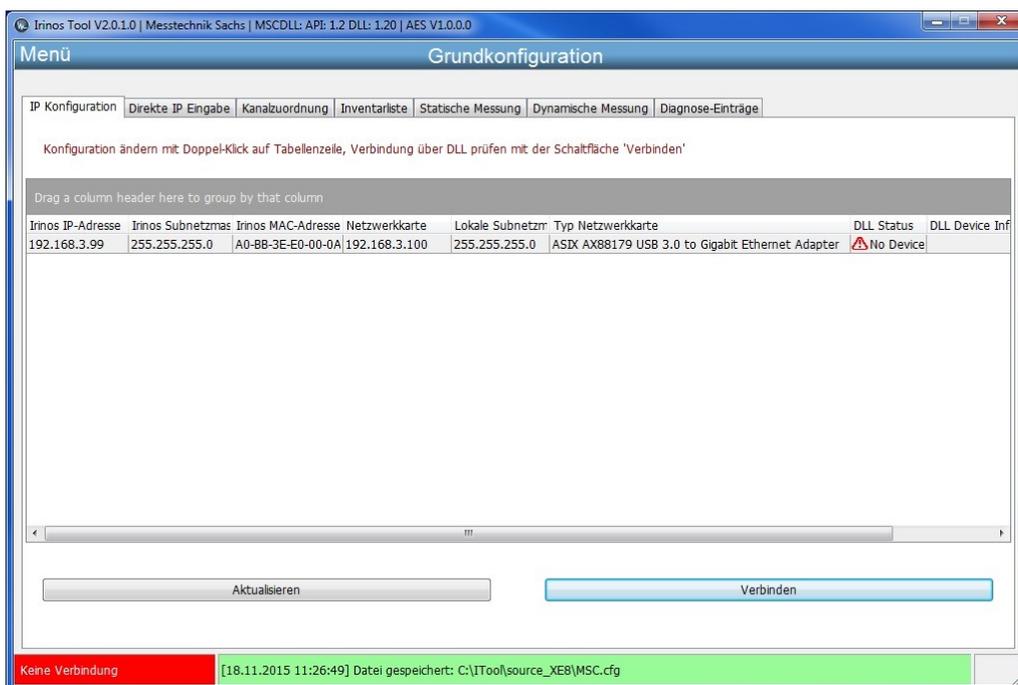
Die Verbindungsprüfung selbst wird, sofern mehrere Systeme vorhanden sein sollten, durch die Auswahl eines Irinos-Systems im Startfenster und Drücken der Schaltfläche „Verbinden“ gestartet. Bei einem erfolgreichen Verbindungsaufbau wird in der Spalte „DLL Status“ ein grünes Häkchen und die Information „Connect“ angezeigt:



Zusätzlich werden in der unteren Statuszeile der Verbindungsstatus als grünes Feld mit Angabe der IP-Adresse des Irinos-Systems angezeigt. Das nächste Feld der Statuszeile enthält den Ablageort der Datei Msc.cfg, mit Speicherdatum und Uhrzeit.

Vom dort angegebenen Pfad kann die Datei bei Bedarf kopiert und in den von der Messrechner-Software benötigten Pfad kopiert werden.

Falls die Verbindungsprüfung fehlschlägt, enthält der DLL-Status neben einem Fehlersymbol die Fehlerinformation, wie sie von der DLL geliefert wird:



Die untere Statuszeile zeigt als Verbindungsstatus „Keine Verbindung“. Unabhängig vom Ergebnis der Verbindungsprüfung wird der Ablageort der Datei MSC.cfg mit Speicherdatum angezeigt.

Als mögliche Ursachen für eine fehlgeschlagene Verbindungsprüfung über die DLL kommen in Betracht:

- Keine physikalische Verbindung zum Irinos-System (Ethernet-Kabel gesteckt ?)
- Netzwerkkarte/-adapter deaktiviert
- [IP-Konfiguration](#)^[110] fehlerhaft
- Keine Spannungsversorgung des Irinos-Systems

3.5.7 Kanalzuordnung

Die Funktion „Kanalzuordnung“ des Irinos-Tools fragt die Messkanalstruktur des Irinos-Systems ab und stellt sie dar. Die Kanalzuordnung wird beim Systemstart in der Master-Box erzeugt und bildet die Verbindung der einzelnen Boxen und ihrer Kanäle innerhalb des Irinos-Systems ab.

Die Darstellung der Kanalzuordnung bildet die Basis für zwei Teilfunktionen:

- Zum einen ermöglicht sie die Änderung des [Eingangstyps bei Inkrementalgebern](#)^[121].
- Zum anderen erlaubt sie die Kanalzuordnung, wie sie von der Master-Box erzeugt wird, als Textdatei zu speichern. Messrechner-Software, die mit früheren Systemen arbeitete, könnte diese Textdatei benötigen, um die [Kanalzuordnung zu ändern](#)^[122]. Für Neuanwendungen ist dies nicht mehr erforderlich.

3.5.7.1 Auswahl der Eingangstyps bei Inkrementalgebern

Inkrementalgeber werden abhängig vom Signaltyp an ihren Ausgängen charakterisiert. Zu den am häufigsten eingesetzten Gebertypen gehören:

- Inkrementalgeber, die sinusförmige 1-Vss-Signale liefern (bezeichnet als Typ „1Vss“)
- Inkrementalgeber, die rechteckförmige Signale mit TTL-/RS422-Pegel liefern (bezeichnet als Typ „TTL“)

Entsprechend dem Gebertyp muss die Irinos-Box, an welche diese Geber angeschlossen werden, vorkonfiguriert werden. Im Auslieferungszustand einer

Inkrementalgeber-Box IR-INC sind alle Kanäle auf den gleichen Gebertyp (1Vss oder TTL, abhängig von der Bestellung) vorkonfiguriert.

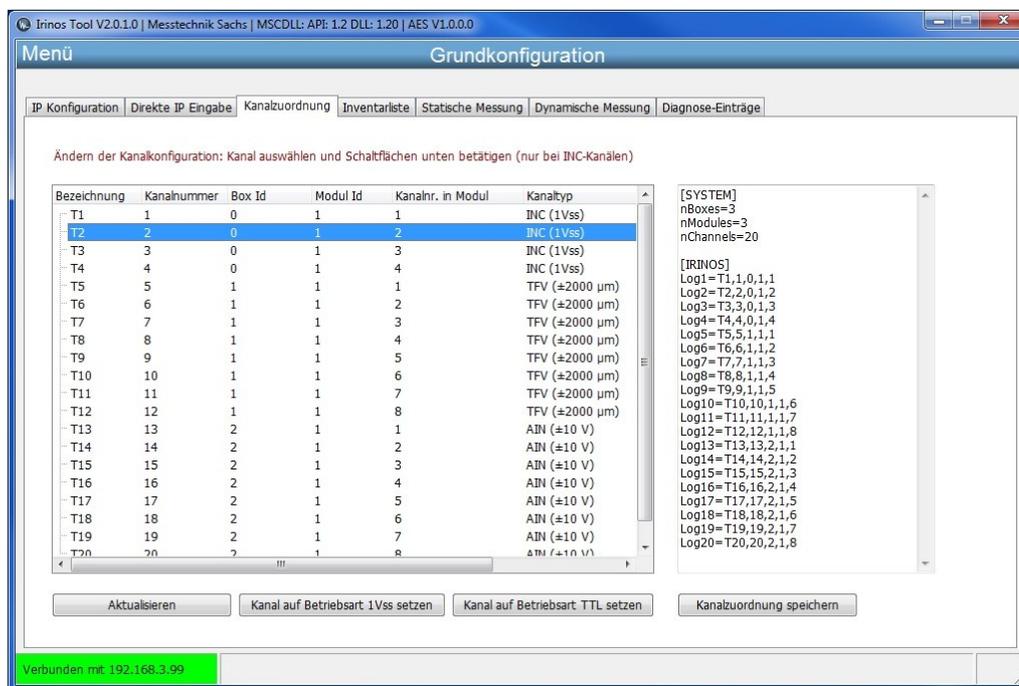
Soll nun diese Vorkonfiguration an Kundenanforderungen angepasst werden, die z.B. einen Mischbetrieb von 1Vss – und TTL-Gebern vorsehen, so kann dies mit dem Irinos-Tool durchgeführt werden.

Die grafische Darstellung der Kanalzuordnung auf der linken Seite des Fensters ermöglicht die Umschaltung dieser Kanalcharakteristik.

Hierzu reicht es aus, einen Kanal des Kanaltyps „INC“ mit der linken Maustaste auszuwählen und eine der Schaltflächen

- Kanal auf Betriebsart 1Vss setzen
- Kanal auf Betriebsart TTL setzen

zu betätigen. Die Änderung des Kanaltyps wird automatisch aktualisiert.



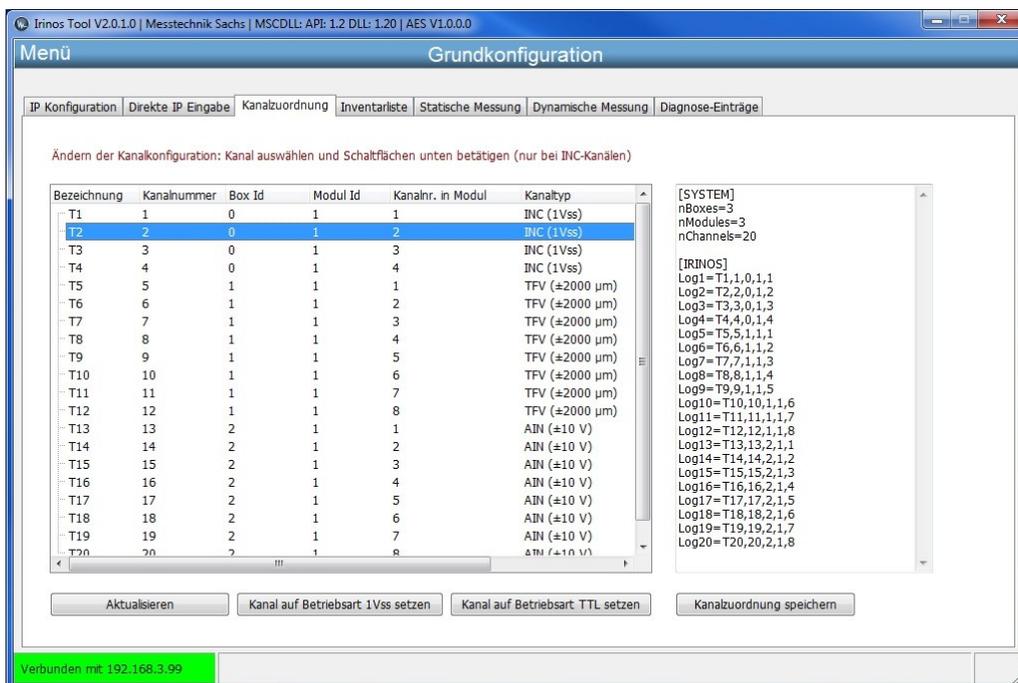
3.5.7.2 Kanalzuordnung ändern

Das Ändern der Kanalzuordnung ist nur zur Kompatibilität mit früheren Systemen implementiert. Für Neu-Applikationen ist dies in der Regel nicht erforderlich.

Die Kanalzuordnung, wie sie vom Irinos-System geliefert wird, bildet die Kanalstruktur des Gesamtsystems ab. Jeder physikalische Messkanal ist dabei einer logischen Kanalnummer zugeordnet. Die logische Kanalnummer muss dabei fortlaufend sein.

Jeder Eintrag der Kanaluordnung besteht somit aus diesen 5 Elementen:

- T1,1,0,1,1** [a] Kanalbeschreibung (ASCII-String, max. 4 Zeichen)
- T1,1,0,1,1** [b] Logische Kanalnummer
- T1,1,0,1,1** [c] Box-ID
- T1,1,0,1,1** [d] Modul-ID. (Immer 1, aus Kompatibilitätsgründen mit früheren Systemen).
- T1,1,0,1,1** [e] Physikalische Kanalnummer innerhalb der Box (aus [c]).



Nach dem Systemstart sind alle Kanäle aktiv und liefern Messdaten im Rahmen der statischen Messung.

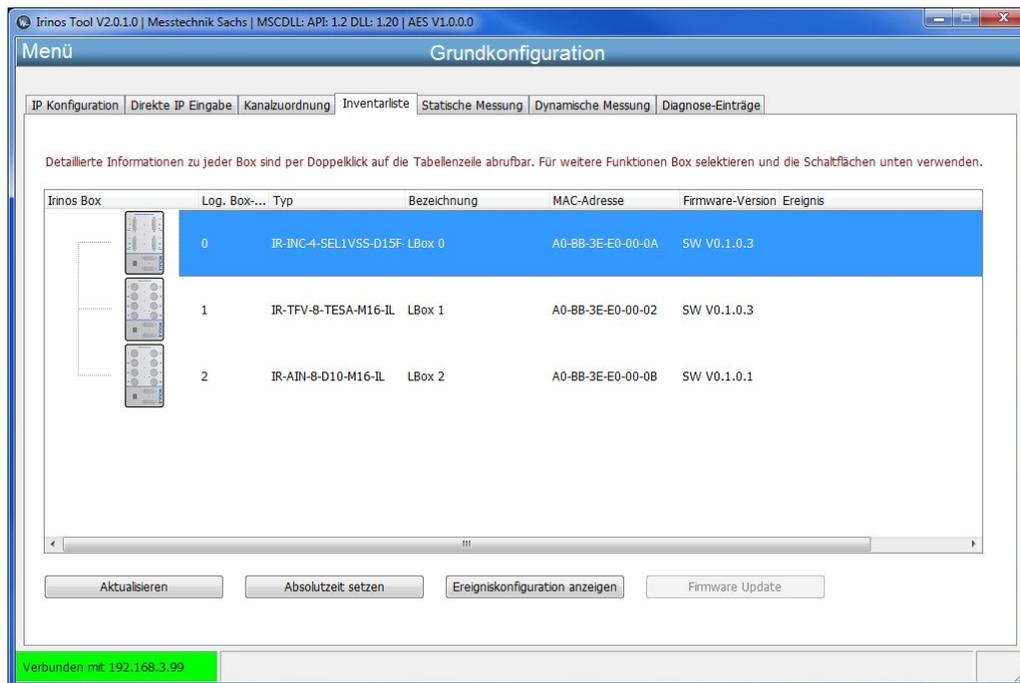
Die Kanaluordnung kann als Textdatei gespeichert, ggf. modifiziert und der Messrechner-Software zur Verfügung gestellt werden, die sie dann wiederum zum Irinos-System sendet.

3.5.8 Inventardaten

Die Inventardatenfunktion des Irinos-Tools liest die Inventardaten aus dem Irinos-System aus und stellt den strukturellen Aufbau des Irinos-Systems grafisch dar.

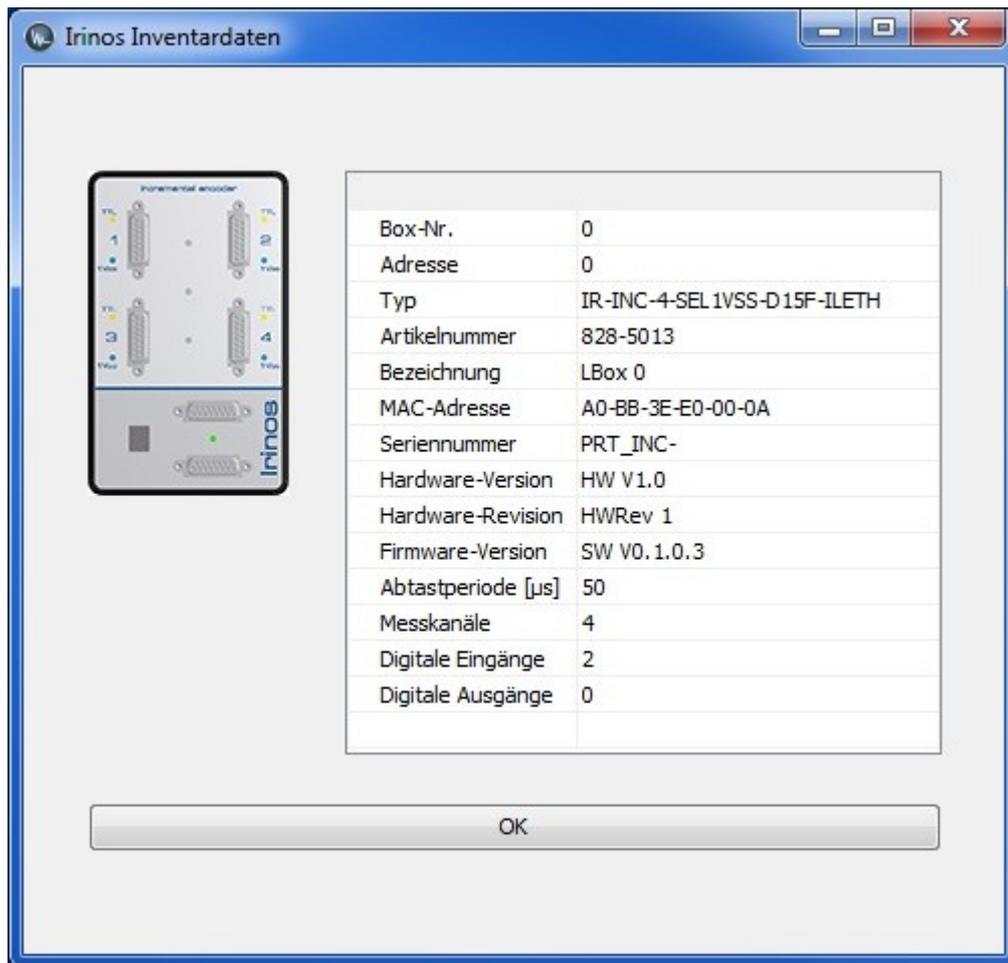
Zusätzlich werden folgende Daten zu jeder Irinos-Box abgerufen und angezeigt:

- Logische Box-Nummer
- Box-Typ
- Box-Bezeichner (wird automatisch vergeben)
- MAC-Adresse
- Firmware-Version
- Aktuelles Ereignis



Zum Aktualisieren der Anzeige wird die Schaltfläche „Aktualisieren“ verwendet. Veränderung am Systemaufbau wie das Hinzufügen oder Entfernen von Irinos-Boxen erfordern jedoch einen Neustart des Irinos-Systems, bevor sie hier dargestellt werden können.

Zusätzlich zu den oben genannten Daten einer Irinos-Box kann per Doppelklick auf die jeweilige Zeile eine umfangreichere Auflistung der Daten einer einzelnen Box angefordert werden:



Neben der Anzeige der Inventardaten dient die Darstellung des Systemaufbaus als Ausgangspunkt für folgende weitere Funktionen:

- [Setzen der Absolutzeit](#)^[125]
- Anzeige der [Ereigniskonfiguration](#)^[126]
- [Firmware-Updates](#)^[132]

Der Zugriff auf diese Funktionen erfolgt, indem eine Irinos-Box über einen linken Mausklick ausgewählt und die entsprechende Schaltfläche betätigt wird.

3.5.8.1 Absolutzeit setzen

--> Die Funktion zum Setzen der Absolutzeit ist nur auf der Master-Box verfügbar.

Über diese Funktion kann das aktuelle Datum und die aktuelle Uhrzeit zum Irinos-System übertragen werden. Das Irinos-System erzeugt daraus eine Referenzzeit, die im Weiteren dazu verwendet wird, auftretende Ereignisse mit einem Zeitstempel auf der Basis von Datum und Uhrzeit auszustatten.

Damit wird eine genaue Zeitkorrelation zwischen dem Auftreten des Ereignisses und der Tageszeit möglich.

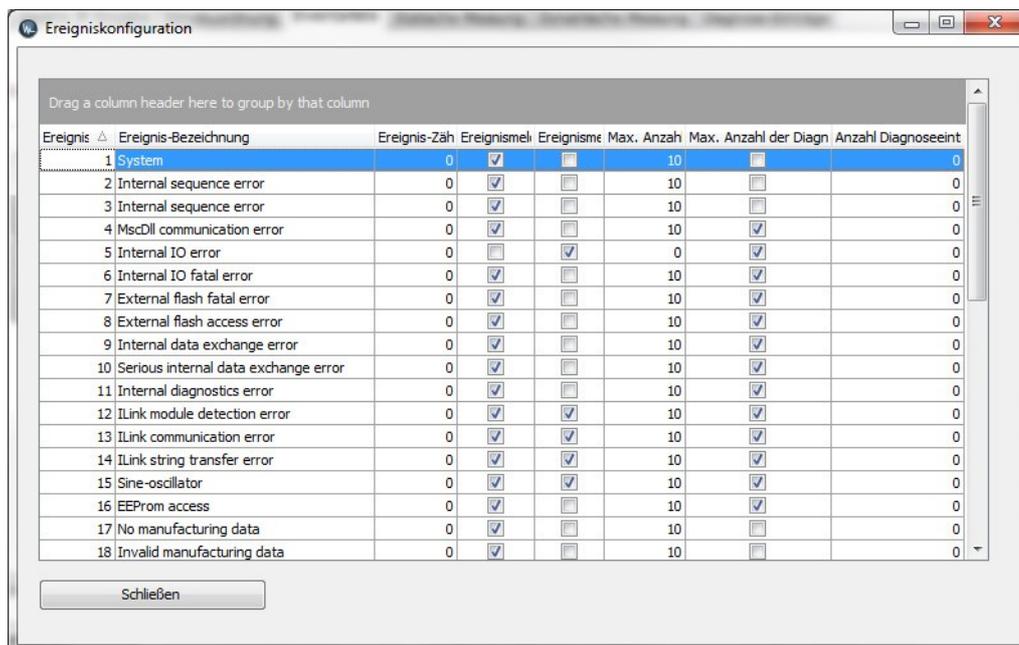
Das Setzen der Absolutzeit muss nach jedem Neustart bzw. Reset des Irinos-System erneut erfolgen.

Im Irinos-Tool dient diese Funktion lediglich als Test- und Analysehilfsmittel. Wird diese Funktion in einer Messrechner-Software eingesetzt, so ist es empfehlenswert, die Zeit einmal pro Tag zu setzen, um eine ausreichende Genauigkeit der Systemzeit zu gewährleisten.

Details zum Aufruf der Funktion und deren Parametrisierung sind im MscDll Referenz-Handbuch beschrieben.

3.5.8.2 Ereigniskonfiguration

Das Fenster "Ereigniskonfiguration" zeigt die verschiedenen Ereignis-Typen und deren Konfigurationsmöglichkeiten im Hinblick auf Weiterverarbeitung und Speicherung. Das Verhalten mancher Ereignistypen kann vom Anwender verändert werden. Folgende Attribute kennzeichnen die Ereignisverwaltung für jeden Ereignistyp:



Drag a column header here to group by that column

Ereignis	Ereignis-Bezeichnung	Ereignis-Zähl	Ereignismel	Ereignism	Max. Anzahl	Max. Anzahl der Diagn	Anzahl Diagnoseint
1	System	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	10	<input type="checkbox"/>	0
2	Internal sequence error	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	10	<input type="checkbox"/>	0
3	Internal sequence error	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	10	<input type="checkbox"/>	0
4	MscDll communication error	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	10	<input checked="" type="checkbox"/>	0
5	Internal IO error	0	<input type="checkbox"/>	<input checked="" type="checkbox"/>	0	<input checked="" type="checkbox"/>	0
6	Internal IO fatal error	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	10	<input checked="" type="checkbox"/>	0
7	External flash fatal error	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	10	<input checked="" type="checkbox"/>	0
8	External flash access error	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	10	<input checked="" type="checkbox"/>	0
9	Internal data exchange error	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	10	<input checked="" type="checkbox"/>	0
10	Serious internal data exchange error	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	10	<input checked="" type="checkbox"/>	0
11	Internal diagnostics error	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	10	<input checked="" type="checkbox"/>	0
12	ILink module detection error	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10	<input checked="" type="checkbox"/>	0
13	ILink communication error	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10	<input checked="" type="checkbox"/>	0
14	ILink string transfer error	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10	<input checked="" type="checkbox"/>	0
15	Sine-oscillator	0	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	10	<input checked="" type="checkbox"/>	0
16	EEProm access	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	10	<input checked="" type="checkbox"/>	0
17	No manufacturing data	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	10	<input type="checkbox"/>	0
18	Invalid manufacturing data	0	<input checked="" type="checkbox"/>	<input type="checkbox"/>	10	<input type="checkbox"/>	0

Schließen

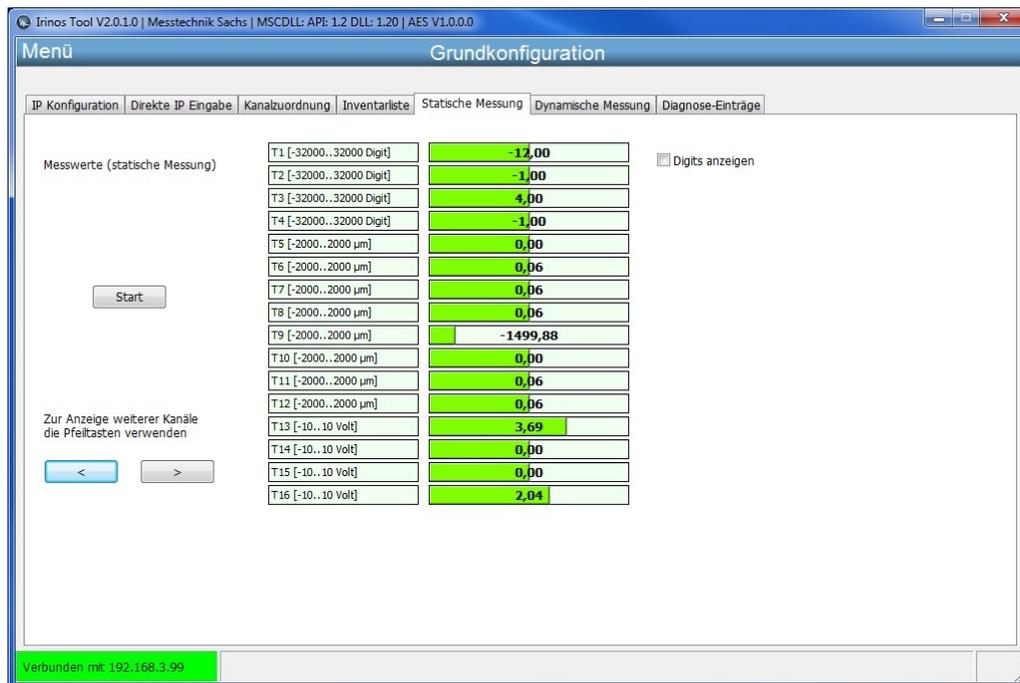
Ereignis-Zähler	Anzahl der aufgetretenen Ereignisse
Ereignismeldung aktiviert	Ereignis wird beim Auftreten als solches behandelt, angezeigt und gespeichert.
Ereignismeldung deaktivierbar	Dieser Ereignistyp kann vom Anwender konfiguriert werden, ob er aktiviert oder deaktiviert sein soll.
Max. Anzahl Diagnose-Einträge	Maximale Anzahl an Diagnose-Einträgen, die von diesem Ereignistyp im Diagnosespeicher gespeichert werden.
Max. Anzahl Diagnose-Einträge änderbar	Die Max. Anzahl an Diagnose-Einträgen kann vom Anwender verändert werden.
Anzahl Diagnose-Einträge	Anzahl tatsächlich gespeicherter Einträge dieses Typs seit Systemstart.

Das Irinos-Tool dient nur zur Anzeige dieser Werte. Um diese Werte zu ändern, werden Funktionen der MscDII verwendet. Details zum Aufruf der Funktion und deren Parametrisierung sind im MscDII Referenz-Handbuch beschrieben.

3.5.9 Statische Messung

Das Irinos-Tool bietet eine Messwertanzeige zur Darstellung und zur Analyse der Messdaten aller Messkanäle. Sobald die statische Messung gestartet wird, erfolgt die Anzeige der „Live“-Daten der Messkanäle. Das Irinos-Tool zeigt 16 Messkanäle gleichzeitig an. Umfasst das Messsystem mehr als 16 Messkanäle, so ist es möglich, mit den Pfeiltasten zwischen den Messkanalgruppen umzuschalten.

Die Umschaltung der Messwertanzeige von den physikalischen Einheiten (μm , Volt, etc.) auf Digits ist über das Kontrollkästchen „Digits anzeigen“ möglich.



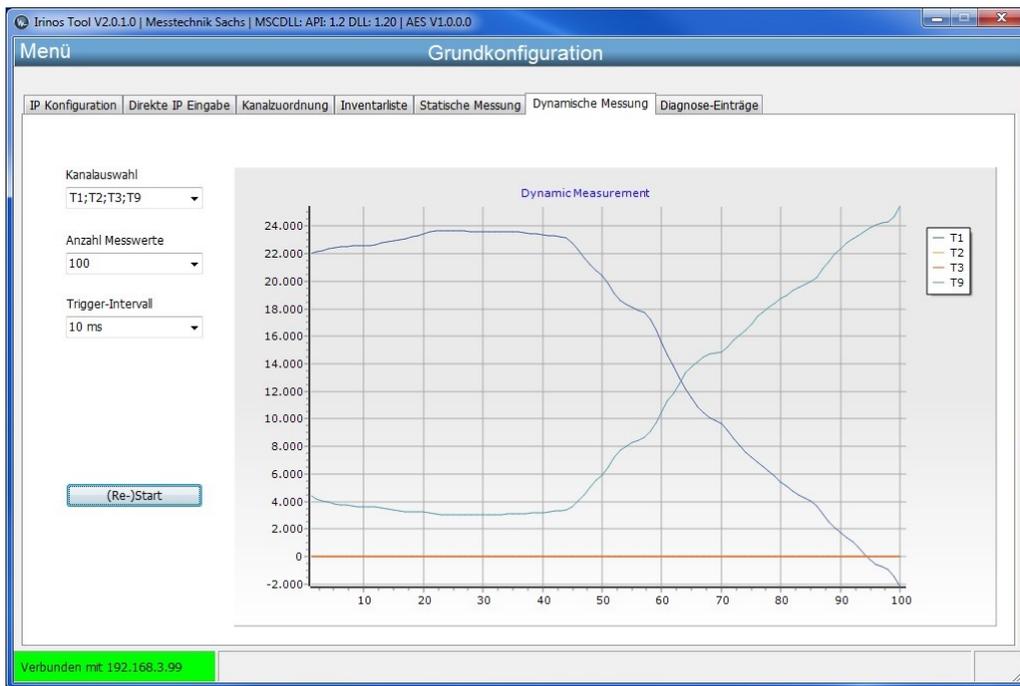
Die statische Messung wird im Zuge der [Verbindungsprüfung](#)¹¹⁹ automatisch gestartet. Lediglich im Fall, dass direkt das Fenster der statischen Messung geöffnet wird, kann es erforderlich sein, die Messung über die Schaltfläche „Start“ zu aktivieren.

3.5.10 Dynamische Messung

Analog zur statischen Messung liefert das Irinos-Tool auch eine Testfunktion zur Ansteuerung der dynamischen Messung. Um die Bedienung einfach zu halten, werden nur dynamische Messungen mit Zeit-Trigger unterstützt.

Um eine dynamische Messung zu konfigurieren, genügt es, per Kanalauswahl die entsprechenden Messkanäle auszuwählen und die Anzahl der Messwerte sowie das Trigger-Intervall vorzugeben. Dann kann die Messung über die Schaltfläche „(Re-)Start“ gestartet werden.

Das Auswahlelement zur Kanalauswahl ist mit Werten vorbelegt, um eine einfache und schnelle Bedienung zur ermöglichen. Es ist jedoch möglich, Kanäle hinzuzufügen oder zu entfernen, solange die Kanalbeschreibung (z.B. T8) im Irinos-System bekannt ist und die strich-punktierte Notation (z.B. T1;T4;T7;T25) beibehalten wird.



3.5.11 Digitale Ein-/Ausgänge

Erfordert die IrinosTool-Version 2.0.1.7 oder neuer.

Über den Bereich Digital-I/O kann der Zustand von 128 digitalen Eingängen angezeigt von von 128 digitalen Ausgängen geändert werden:

The screenshot shows the 'Digital-I/O' configuration window in Irinos Tool V2.0.1.7. It features two tables for configuring digital inputs and outputs. The 'Input' table has columns for Byte 1 through Byte 16 and rows for Bit 1 through Bit 8. The 'Output' table has the same structure. Checkmarks in the 'Bit 1' row of both tables indicate that bit 1 is active for both input and output. A 'Start Bit-I/O' button is located at the bottom. The status bar at the bottom shows 'Verbunden mit 192.168.3.99' and a file save path: 'C:\ITool\source_XE8\MSC.cfg'.

Input	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15	Byte 16
Bit 1	<input checked="" type="checkbox"/>	<input type="checkbox"/>														
Bit 2	<input checked="" type="checkbox"/>	<input type="checkbox"/>														
Bit 3	<input checked="" type="checkbox"/>	<input type="checkbox"/>														
Bit 4	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Bit 5	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Bit 6	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Bit 7	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Bit 8	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Output	Byte 1	Byte 2	Byte 3	Byte 4	Byte 5	Byte 6	Byte 7	Byte 8	Byte 9	Byte 10	Byte 11	Byte 12	Byte 13	Byte 14	Byte 15	Byte 16
Bit 1	<input checked="" type="checkbox"/>	<input type="checkbox"/>														
Bit 2	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Bit 3	<input checked="" type="checkbox"/>	<input type="checkbox"/>														
Bit 4	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Bit 5	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Bit 6	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Bit 7	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
Bit 8	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Digitale Ein-/Ausgänge

Die Tabelle „Input“ zeigt den Status der angeschlossenen Eingänge des Irinos-Systems an. Ein Häkchen in der Checkbox kennzeichnet dabei den High-Pegel an einem digitalen Eingang.

Mit Hilfe der Tabelle „Output“ können Ausgänge aktiv durch den Anwender geschaltet werden. Setzt man ein Häkchen in eine Checkbox, so wird der entsprechende Ausgang der Irinos-Box bzw. der angeschlossenen I/O-Box auf High-Pegel gesetzt.

3.5.12 Diagnose-Speicher

Für Diagnosezwecke sammelt und speichert das Irinos-System Ereignisse, die von der Firmware gemeldet werden, in einem nicht-flüchtigen Diagnose-Speicher. Damit sind sie auch nach einem System-Neustart verfügbar und können zur Analyse ausgelesen werden.

Das Auslesen kann sowohl über den integrierten Web-Server, als auch durch Abfrage der Daten über das Irinos-Tool erfolgen. Ereignisse aus dem Diagnose-Speicher werden im Irinos-Tool für das gesamte Irinos-System dargestellt. Dies wird ermöglicht, in dem nacheinander die Daten der einzelnen Boxen abgefragt werden und die Ereignisdaten dann in einer gemeinsamen Tabelle dargestellt werden. Gruppier- und Sortierfunktionen der Tabelle erlauben damit auch Box-übergreifende Analysen.

Box-Nr.	Nr.	Ereignis-	Ereignis-Be	Zusatzinfo	Absolutzeit	Interne Zeit	Firmware-Version
0	1	1 System		System started	0000-00-00 00:00:00:0	0	V0.4.0.4
0	2	28 Firmware u		Firmware-Update successfully finished.	0000-00-00 00:00:00:0	0	V0.4.0.4
0	3	1 System		System started	0000-00-00 00:00:00:0	0	V0.4.0.3
0	4	28 Firmware u		Firmware-Update successfully finished.	0000-00-00 00:00:00:0	0	V0.4.0.3
0	5	1 System		System started	0000-00-00 00:00:00:0	0	V0.4.0.1
0	6	1 System		Diag memory cleared	0000-00-00 00:00:00:0	324582150	V0.4.0.1
1	1	15 Sine-oscillat		Probe short circuit.	2016-01-29 18:43:19:354	45417980	V0.4.0.1
1	2	1 System		System started	0000-00-00 00:00:00:0	0	V0.4.0.1
1	3	15 Sine-oscillat		Probe short circuit.	2016-01-08 12:04:10:663	10845360	V0.4.0.1
1	4	1 System		System started	0000-00-00 00:00:00:0	0	V0.4.0.1
1	5	1 System		Diag memory cleared	0000-00-00 00:00:00:0	319085180	V0.4.0.1

Ein Ereignis ist durch folgende Attribute gekennzeichnet:

Box-Nr	Nummer der Box, in der das Ereignis aufgetreten ist
Nr	Ereignisnummer (pro Box)
Ereignistyp	Ereignistyp als numerischer Wert
Ereignisbeschreibung	Ereignistyp als Text
Zusatzinfo	Ergänzende Informationen zur Ereignis-Ursache
Absolutzeit	Auftrittszeitpunkt als Datum-Uhrzeit-Format (sofern die Absolut-Zeit ¹²⁵ zuvor gesetzt wurde).
Interne Zeit	Interne Systemzeit (ILink-Zeit, [µs])
Firmware-Version	Firmware-Version der Box zum Ereigniszeitpunkt
Debug-Info	Erweiterte Informationen für den Support

Neu hinzukommende Ereignisse können über die Schaltfläche „Aktualisieren“ abgefragt werden.

Mit der Schaltfläche "Als CSV-Datei speichern" können sämtliche Diagnose-Einträge in einer CSV-Datei abgelegt werden. Über einen Benutzer-Dialog können Verzeichnis und Dateiname vorgegeben werden.

3.5.13 Firmware-Update

3.5.13.1 Versionsnummern

Die Versionsnummer der Firmware besteht immer aus 4 Teilen, die durch einen Punkt voneinander getrennt sind, z.B. V1.3.4.534. Die jeweilige Bedeutung der einzelnen ist:

Teil der Versionsnummer	Bedeutung
1. Teil, im Beispiel: 1	Hauptversionsnummer ("Major") Diese wird bei einer umfassenden Überarbeitung der Firmware inkrementiert, in der Praxis also sehr selten.
2. Teil, im Beispiel: 3	Nebenversionsnummer ("Minor") Diese wird bei jeder funktionalen Änderung/Erweiterung der Firmware inkrementiert.
3. Teil, im Beispiel: 4	Revisionsnummer ("Patch") Diese wird bei jeder Änderung inkrementiert, die keine funktionale Auswirkung hat (z.B. Fehlerbehebung).
4. Teil, im Beispiel: 534	Buildnummer ("Build") Interne Nummer zur Kennzeichnung des Zeitpunktes der Firmware-Erzeugung.

Die Zusammensetzung der Versionsnummer entspricht damit einer weit verbreiteten Vorgehensweise.

Eine Firmware kann darüber hinaus als "kundenspezifisch" oder als "Beta-Version" gekennzeichnet sein ("Firmware-Typ"). Diese Firmware darf nur nach schriftlicher Freigabe durch Messtechnik Sachs verwendet werden. Ein entsprechender Warnhinweis erscheint, bevor die Firmware zum Irinos-System übertragen wird.

3.5.13.2 Update ausführen

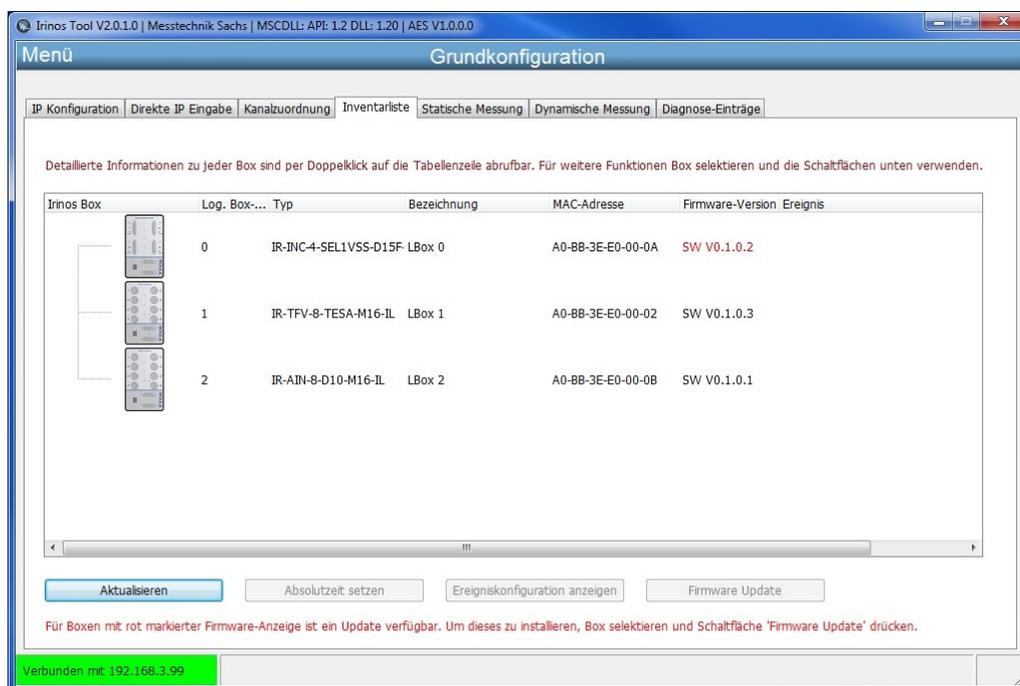
Die Unterstützung von Firmware-Updates zählt zu den grundlegenden Aufgaben des Irinos-Tools.

Firmware-Pakete werden als spezielle Datei mit der Dateierweiterung `*.SFF` zur Verfügung gestellt. Firmware-Pakete sind spezifisch für den jeweiligen Boxentyp. Das bedeutet, dass ein Firmware-Paket, das für eine Inkrementalgeberbox herausgegeben wurde nur für diesen Boxentyp verwendet werden kann.

Während der Installation des Irinos-Tools wird bereits ein Dateiordner „Firmware“ im Verzeichnis „MeinOrdner/ITool/source_XEn“ angelegt. **Zu installierende Firmware-Pakete müssen in diesen Ordner „Firmware“ abgelegt werden, damit sie vom Irinos-Tool gefunden werden.**

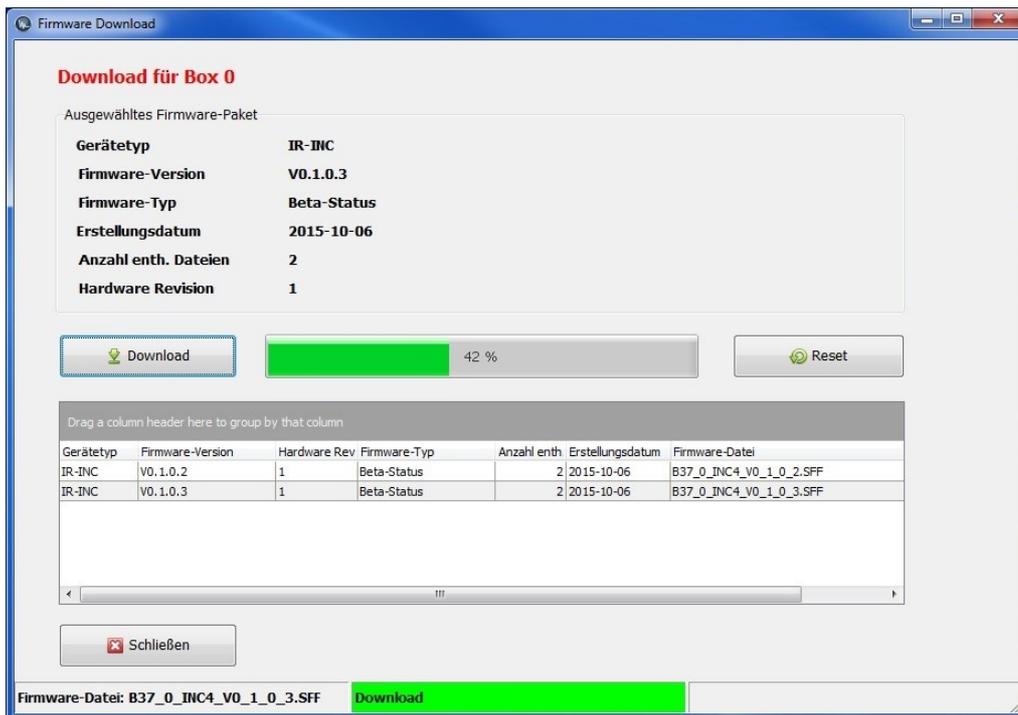
Das Irinos-Tool liest und überprüft die Firmware-Pakete während der Startphase. Kommen Firmware-Pakete dazu, so ist ein Neustart des Irinos-Tools erforderlich.

Wird ein neues Firmware-Paket erkannt und einem Boxentyp zugeordnet, so wird in der Inventardatenanzeige die aktuelle Firmware-Version der Box rot markiert. Zusätzlich wird am unteren Rand des Fensters ein Hinweis angezeigt, dass für die markierten Boxen ein Firmware-Update möglich ist:

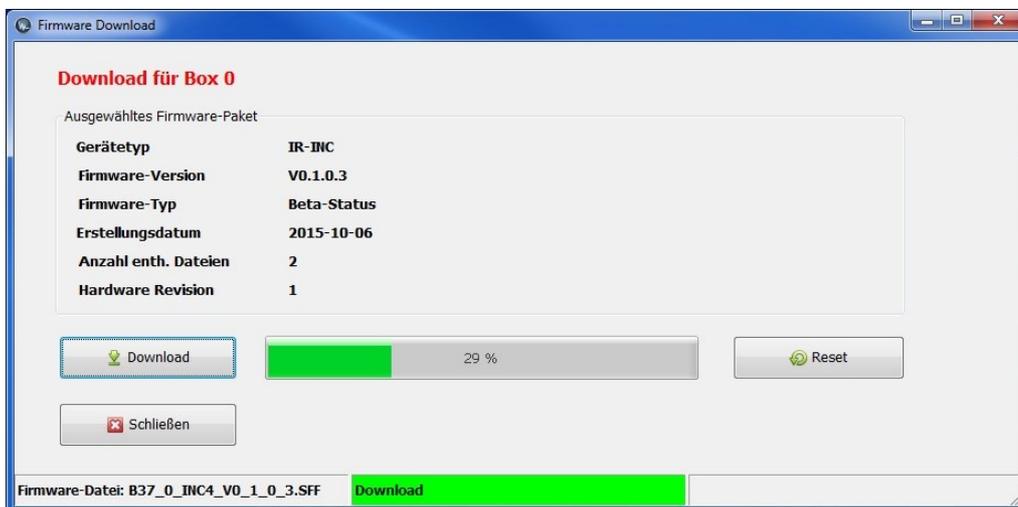


Wird eine Box per Mausklick ausgewählt, so öffnet ein weiterer Klick auf die Schaltfläche „Firmware Update“ das entsprechende Update-Fenster.

Werden pro Boxentyp mehrere Firmware-Pakete gefunden, so zeigt das Firmware-Download-Fenster eine Tabelle mit Auswahlmöglichkeiten an. Erst nachdem der Bediener per Mausklick eine Auswahl in der Tabelle getroffen hat, kann der Download über die Schaltfläche „Download“ gestartet werden:



Existiert nur ein Firmware-Paket für diesen Boxentyp, so wird dies automatisch ausgewählt und der Download kann direkt über die Schaltfläche „Download“ gestartet werden:



Wurden alle Firmware-Pakete per Download in die Irinos-Boxen eingespielt, ist ein System-Reset erforderlich um die geladenen Firmware-Pakete zu aktivieren.

Über die Schaltfläche „Reset“ wird ein entsprechendes Kommando an das Irinos-System gesendet. Dabei sorgt eine interne Ablaufsteuerung dafür, dass Slave-Boxen zuerst zurückgesetzt werden, bevor die Masterbox zurückgesetzt

wird. Dies ist notwendig, da eine Masterbox beim Start immer aktive, kommunikationsbereite Slave-Boxen erwartet.

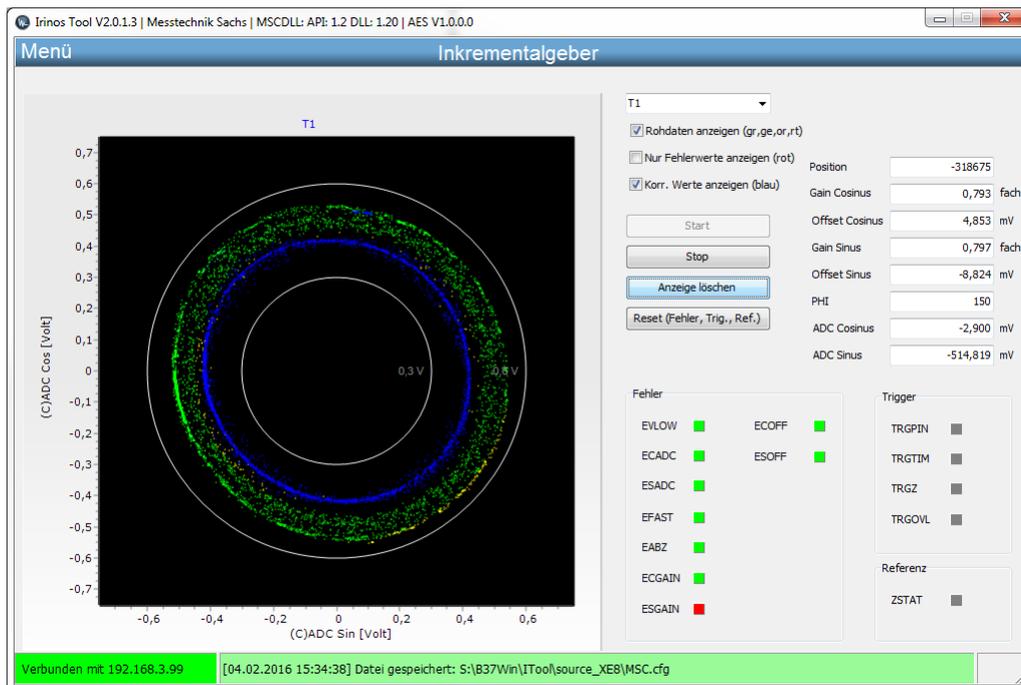
3.5.14 Inkrementalgeber-Diagnose

3.5.14.1 Live-Anzeige (nur 1Vss)

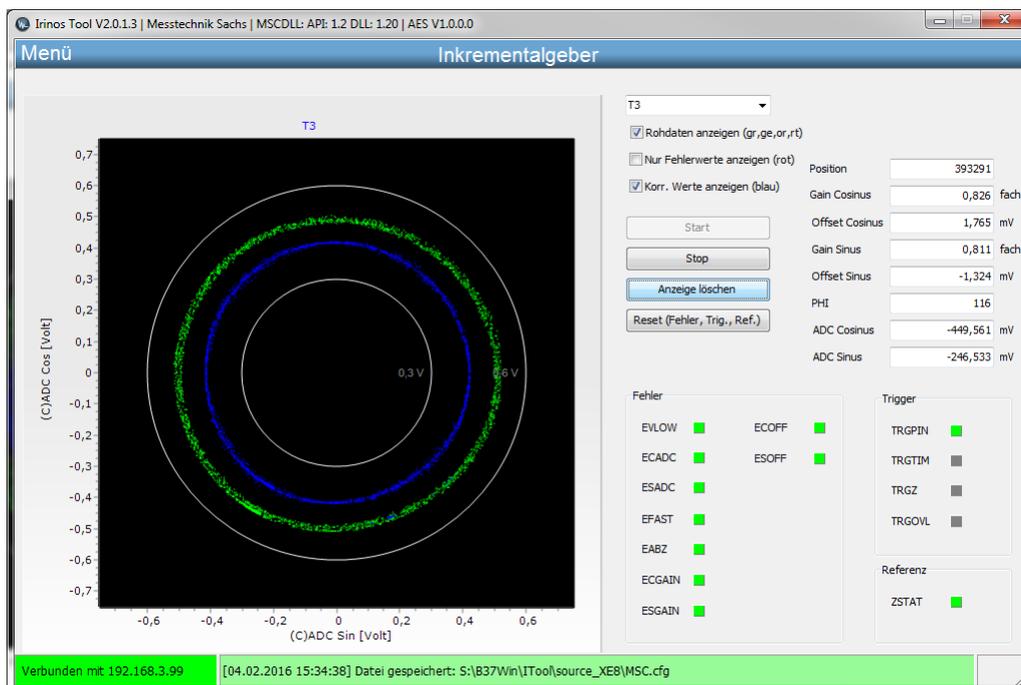
Bei Inkrementalgebern mit 1Vss - Schnittstelle hat die Signalqualität einen großen Einfluss auf die Zuverlässigkeit des Messwertes. Die Inkrementalgeber-Eingänge des Irinos-Systems führen eine sehr ausführliche Signalanalyse durch. Deshalb können vom Irinos-System Fehler erkannt werden, wo viele andere Systeme eine noch vermeintlich korrekte Position liefern.

Liegt die Signalqualität außerhalb des spezifizierten Bereichs, kann nicht sichergestellt werden, dass die gelieferte Position korrekt ist. Der jeweilige Inkrementalgeber-Eingang liefert dann einen Fehler. Mehr dazu entnehmen Sie den Applikationshinweisen im Benutzerhandbuch.

Mit der Live-Anzeige des Irinos-Tools können die Inkrementalgeber-Signale in einem Lissajous-Diagramm dargestellt werden. Dabei werden der Signalpegel des Sinus-Signals in der x-Achse und der zugehörige Signalpegel des Cosinus-Signals in der y-Achse dargestellt. Bei idealen Sinus- und Cosinus-Signalen eines Inkrementalgebers ergibt sich über eine elektrische Umdrehung hinweg ein Einheitskreis bei 0,5 Volt. In der Praxis wird ein derartiges Signal nie erreicht. Folgende Beispiele zeigen die reale Signalqualität für einen Inkrementalgeber mit schlechter Signalqualität bzw. mit akzeptabler Signalqualität:



Live-Anzeige eines Inkrementalgebers mit hoher Streuung der Eingangssignale



Live-Anzeige eines Inkrementalgebers mit akzeptabler Signalqualität

Technisch bedingt haben die Signale am Inkrementalgeber-Eingang nie exakt die spezifizierten $1V_{SS}$ Eingangsspiegel. Deshalb hat die Irinos-Box IR-INC einen Toleranzbereich von $0,6V_{SS}$.. $1,2V_{SS}$, in welchem die Signale liegen dürfen.

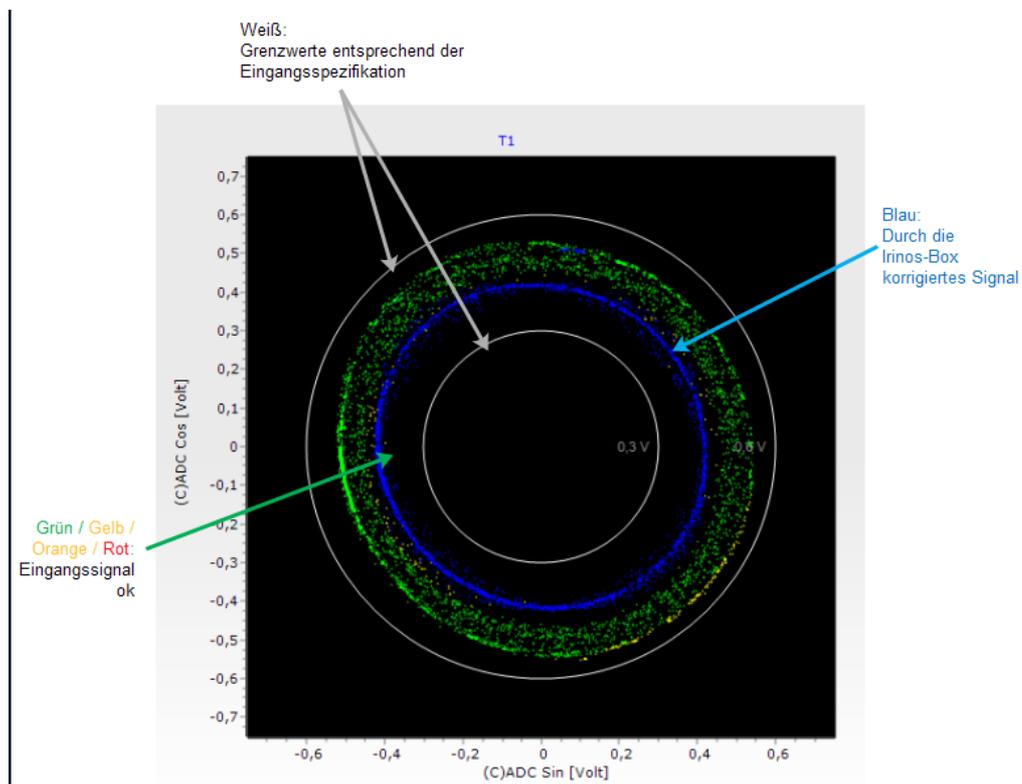
Lissajous-Diagramm

Anhand des Lissajous-Diagramms kann abgeschätzt werden, wie gut die Signalqualität eines Inkrementalgebers ist. Dazu werden durch das Irinos-Tool die Rohwerte eines Inkrementalgeber-Eingangs sowie dazugehörige Zusatzinformationen kontinuierlich abgefragt. Starten Sie eine Live-Anzeige wie folgt:

1. Stellen Sie eine [Verbindung](#) ¹¹⁹ zum Irinos-System her.
2. Öffnen Sie die Live-Anzeige über Menü->Inkrementalgeber.
3. Wählen Sie den zu untersuchenden Eingangs-Kanal aus (z.B. T3).
4. Drücken Sie den Button "Start".
5. Bewegen Sie den Inkrementalgeber.

Die resultierende Signalvektor der Eingangssignale ("Rohdaten") wird nun angezeigt. Liegt der Signalvektor nahe des Einheitskreises, wird er in grün dargestellt. Je weiter er davon weg liegt, wird er in gelb, orange oder rot dargestellt.

Der innere bzw. äußere Grenzwert wird jeweils durch eine weiße Linie dargestellt.



Lissajous-Diagramm

Die Irinos-Box führt eine kontinuierliche Korrektur des Eingangssignals durch, indem die Phase und der Offset der beiden Eingangssignale korrigiert werden. Die korrigierten Werte werden im Lissajous-Diagramm blau dargestellt. Die Korrektur kann einen Großteil der Abweichungen ausgleichen. Sobald die Eingangssignale aber zu schlecht sind, geht auch dies nicht mehr.

Dieses Beispiel Lissajous-Diagramm zeigt eine große Streuung des Eingangssignals. Die meisten Werte sind zwar grün, vereinzelt sind aber auch gelbe und orange Werte erkennbar. Das Diagramm entstand bei moderater Bewegung eines Inkrementalgebers. Bei schnellerer Bewegung ist es wahrscheinlich, dass die Eingangssignale außerhalb des gültigen Bereichs liegen werden.

Zahlenwerte

Rechts neben dem Lissajous-Diagramm werden die Daten auch als Zahlenwerte angezeigt:

Position	<input type="text" value="-318675"/>	
Gain Cosinus	<input type="text" value="0,793"/>	fach
Offset Cosinus	<input type="text" value="4,853"/>	mV
Gain Sinus	<input type="text" value="0,797"/>	fach
Offset Sinus	<input type="text" value="-8,824"/>	mV
PHI	<input type="text" value="150"/>	
ADC Cosinus	<input type="text" value="-2,900"/>	mV
ADC Sinus	<input type="text" value="-514,819"/>	mV

Werteanzeige

Position

Messwert / Positionswert des Inkrementalgeber-Eingangs

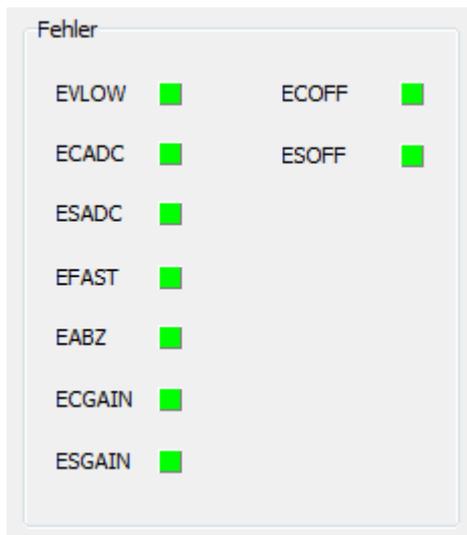
Gain Cosinus

Verstärkungsfaktor für das Cosinus-Signal. Dieser wird durch die Verstärkungsregelung laufend korrigiert.
Er kann auch bei idealem Eingangssignal ungleich 1 sein.

Offset Cosinus	Offset-Korrektur des Cosinus-Signals. Dieser Wert wird durch die Offsetregelung laufend korrigiert.
Gain Sinus	Verstärkungsfaktor für das Sinus-Signal. Dieser wird durch die Verstärkungsregelung laufend korrigiert. Er kann auch bei idealem Eingangssignal ungleich 1 sein.
Offset Sinus	Offset-Korrektur des Sinus-Signals. Dieser Wert wird durch die Offsetregelung laufend korrigiert.
PHI	Phasenwinkel des Eingangssignals. 0 -> 0° 200 -> 360°
ADC Cosinus	Gemessene Analogspannung am Cosinus-Eingang. Bei einem idealen Signal schwankt dieser Wert zwischen -500mV .. +500mV.
ADC Sinus	Gemessene Analogspannung am Sinus-Eingang. Bei einem idealen Signal schwankt dieser Wert zwischen -500mV .. +500mV.

Fehler

In der Fehleranzeige wird der Zustand der einzelnen Fehlerbits angezeigt (grün = ok, rot = Fehler):



Fehleranzeige

Fehler-Flag

EVLOW

EADDC

ESADC

EFAST

Ursache

Der aus Cosinus- und Sinussignal gebildete Signalvektor ist zu klein. Ursache ist meist ein teilweiser bzw. vollständiger Sensorabriss. Für Signale mit sehr großem Offset bei gleichzeitig kleiner Amplitude kann dieser Fehler ebenfalls auftreten.

Der AD-Wandler für das Cosinussignal ist übersteuert. Ursache ist eine zu große Signalamplitude. Für Signale mit sehr großem Offset bei gleichzeitig großer Amplitude kann dieser Fehler ebenfalls auftreten.

Der AD-Wandler für das Sinussignal ist übersteuert. Ursache ist eine zu große Signalamplitude. Für Signale mit sehr großem Offset bei gleichzeitig großer Amplitude kann dieser Fehler ebenfalls auftreten.

Die Eingangsfrequenz ist zu hoch.

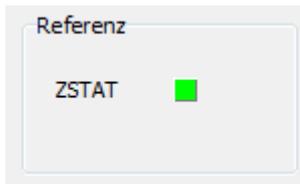
EABZ	Interner Fehlermerker, der für die Standard-Betriebsart deaktiviert ist.
ECGAIN	Der Verstärkungsregler für das Cosinussignal hat seine Grenze erreicht. Ursache ist eine zu kleine Signalamplitude, ein teilweiser oder ein vollständiger Sensorabriss.
ESGAIN	Der Verstärkungsregler für das Sinussignal hat seine Grenze erreicht. Ursache ist eine zu kleine Signalamplitude, ein teilweiser oder ein vollständiger Sensorabriss.
ECOFF	Der Offsetregler für das Cosinussignal hat seine Grenze erreicht. Ursache ist ein zu großer Signaloffset, ein ungültiger Wert zur Initialisierung des Reglers, ein teilweiser oder ein vollständiger Sensorabriss.
ESOFF	Der Offsetregler für das Sinussignal hat seine Grenze erreicht.

Beachten Sie in Bezug auf auftretende Fehler:

Im Lissajous-Diagramm der Live-Anzeige wird nur ein kleiner Teil der für die Auswertung verwendeten Rohwerte genutzt. Es kann deshalb durchaus vorkommen, dass ein Fehler auftritt, jedoch im Lissajous-Diagramm kein zugehöriger Ausreißer sichtbar ist.

Referenzmarke

Das Bit ZSTAT der Referenzanzeige ist aktiv, wenn die Referenzmarke überfahren wurde.



3.5.14.2 Historie (nur 1Vss)

Erfordert die IrinosTool-Version 2.0.1.7 oder neuer.

Bei Inkrementalgebern mit 1Vss-Schnittstelle beeinflusst die Signalqualität des Gebersignals entscheidend die Zuverlässigkeit des Messsystems. Liegt die Signalqualität außerhalb des spezifizierten Bereichs, führt dies unweigerlich zu Fehlern, die von der Auswerte-Logik erkannt und gemeldet werden.

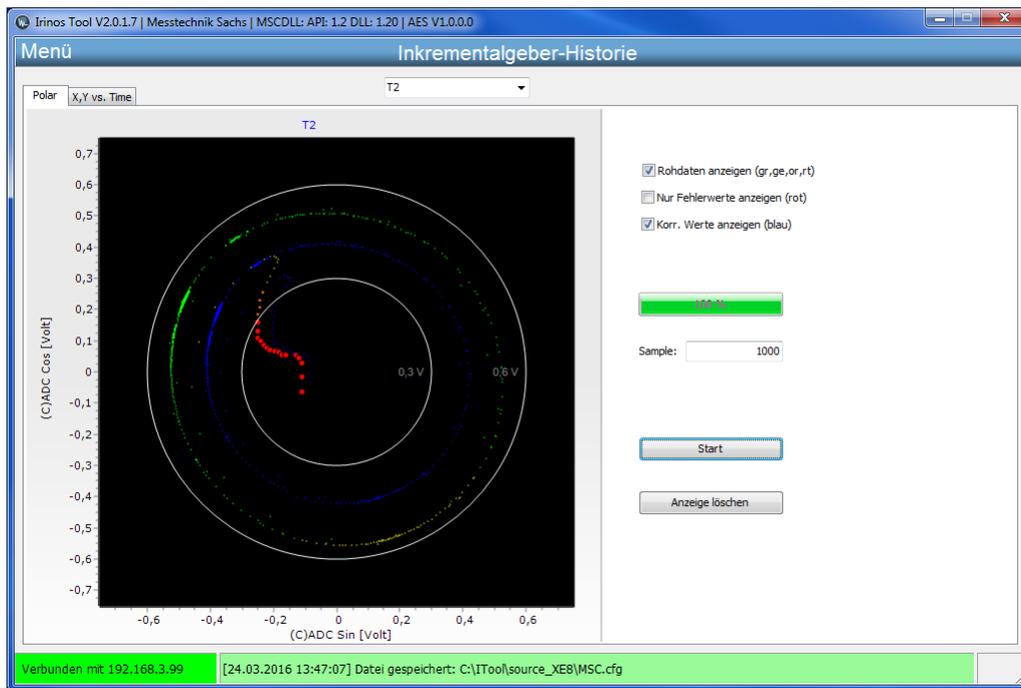
Um nun Rückschlüsse zu ermöglichen, durch welche Einschränkungen der Signalqualität der Fehler ausgelöst wurde, speichert das Irinos-System die letzten 1000 Signalwerte (entspricht 1s) vor dem Auftreten des Fehlers.

Das Irinos-Tool bietet mit der Funktion „Inkrementalgeber-Historie“ die Möglichkeit, diese gespeicherten Werte zu visualisieren und eine nachgelagerte Analyse durchzuführen.

Sowohl die Darstellung als Lissajous-Diagramm, wie sie bei der Live-Anzeige der Daten zum Einsatz kommt, als auch die Darstellung der Werte über der Zeit wird dabei unterstützt.

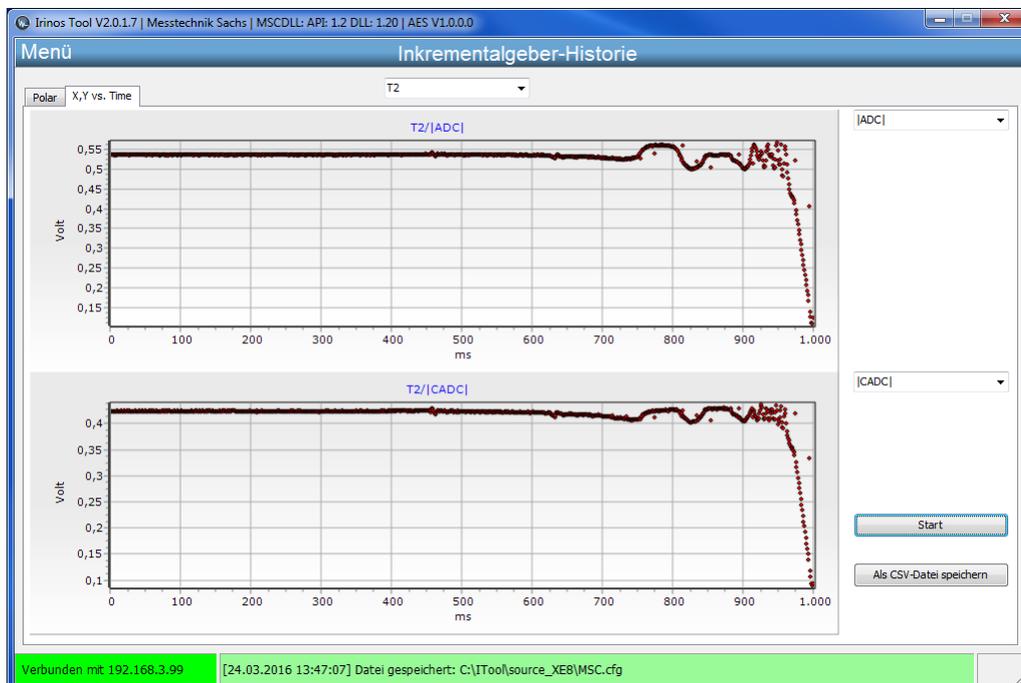
Folgende Abbildung zeigt in der Karteikarte „Polar“ beispielhaft den Signalverlauf über die letzten 1000 Werte vor dem Auftreten des Fehlers. Analog zu der [Live-Anzeige](#)¹³⁵ wird die Signalgüte farblich gekennzeichnet. Grün kennzeichnet Werte mit guter Signalqualität, mit abnehmender Signalqualität verändert sich die Darstellung nach gelb, orange und rot.

Deutlich ist in diesem Beispiel die drastische Verschlechterung der Signalqualität bei den letzten 16 Werten zu erkennen (rot dargestellte Punkte).



Inkrementalgeber-Historie: Lissajous-Diagramm

Eine weitere Möglichkeit ist die Anzeige der gespeicherten Werte über der Zeitachse:



Inkrementalgeber-Historie: Werte über der Zeitachse

Zwei übereinander angeordnete Diagramme ermöglichen zusätzlich eine zeitliche Korrelation verschiedener Werte zueinander. Über Auswahlm

rechts von den Tabellen kann der darzustellende Wert ausgewählt werden. Die Auswahl umfasst folgende Werte:

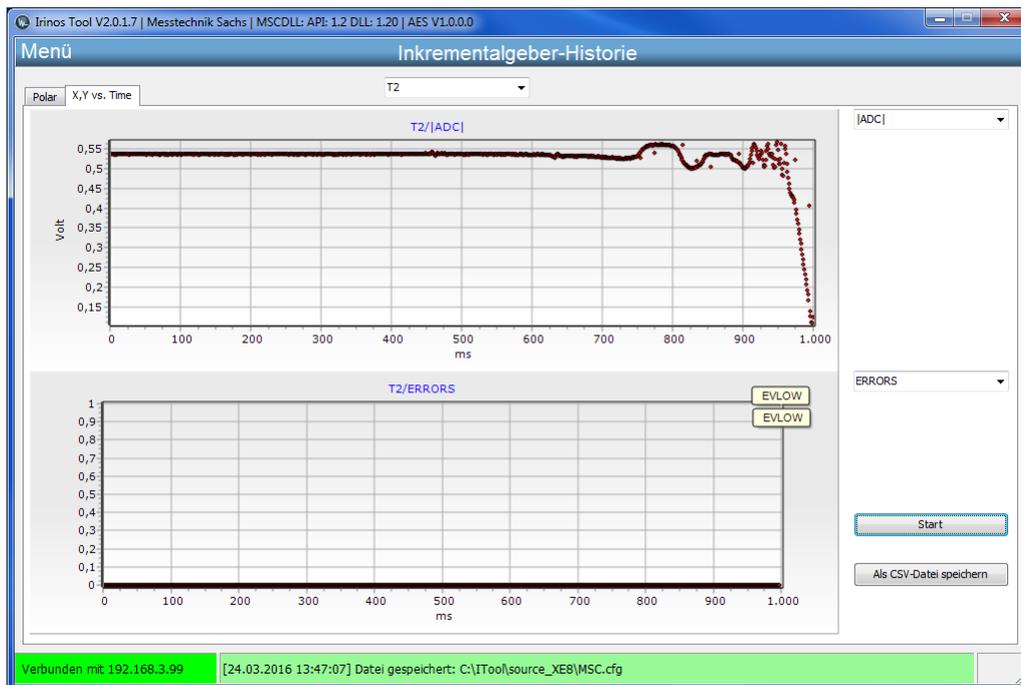
- Betrag des ADC- Signals (entspricht dem Signalvektor aus der Live-Datendarstellung)
- Betrag des korrigierten ADC- Signals (s.o.)
- Sinuswert des ADC-Signals
- Cosinuswert des ADC-Signals
- Korrigierter Sinuswert des ADC-Signals
- Korrigierter Cosinuswert des ADC-Signals

Darüber hinaus kann das Auftreten der einzelnen Fehler-Flags in die Darstellung miteinbezogen werden. Sämtliche unten aufgeführte Fehler-Flags sind daher im Auswahlmenü enthalten:

Fehler-Flag	Ursache
EVLOW	Der aus Cosinus- und Sinussignal gebildete Signalvektor ist zu klein. Ursache ist meist ein teilweiser bzw. vollständiger Sensorabriss. Für Signale mit sehr großem Offset bei gleichzeitig kleiner Amplitude kann dieser Fehler ebenfalls auftreten.
ECADC	Der AD-Wandler für das Cosinussignal ist übersteuert. Ursache ist eine zu große Signalamplitude. Für Signale mit sehr großem Offset bei gleichzeitig großer Amplitude kann dieser Fehler ebenfalls auftreten.
ESACD	Der AD-Wandler für das Sinussignal ist übersteuert. Ursache ist eine zu große Signalamplitude. Für Signale mit sehr großem Offset bei gleichzeitig großer Amplitude kann dieser Fehler ebenfalls auftreten.
EFAST	Die Eingangsfrequenz ist zu hoch.

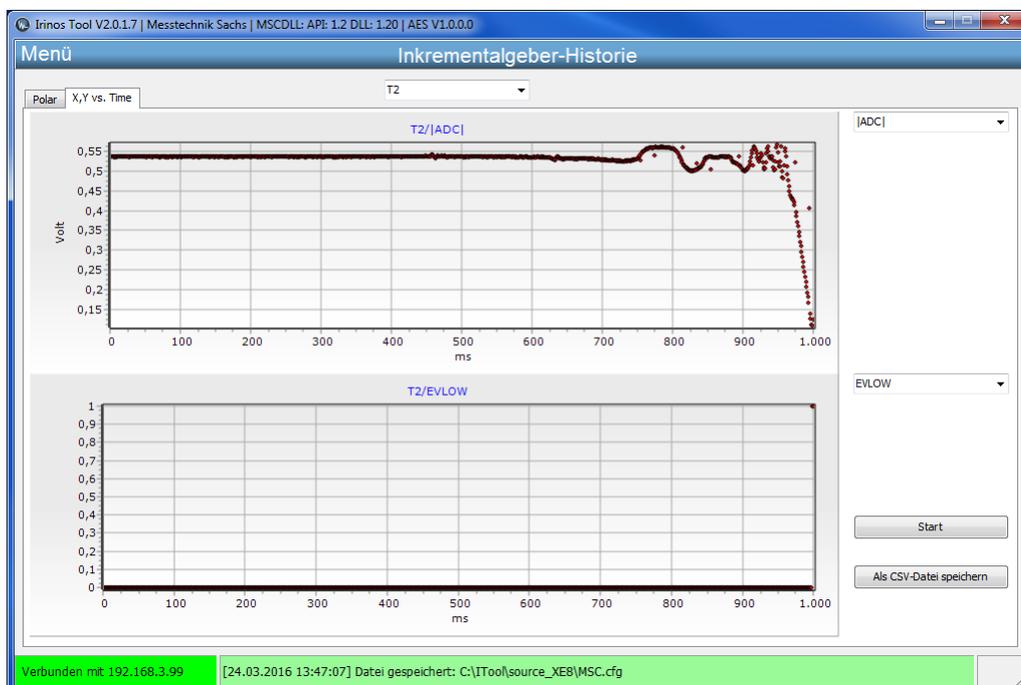
EABZ	Interner Fehlermerker, der für die Standard-Betriebsart deaktiviert ist.
ECGAIN	Der Verstärkungsregler für das Cosinussignal hat seine Grenze erreicht. Ursache ist eine zu kleine Signalamplitude, ein teilweiser oder ein vollständiger Sensorabriss.
ESGAIN	Der Verstärkungsregler für das Sinussignal hat seine Grenze erreicht. Ursache ist eine zu kleine Signalamplitude, ein teilweiser oder ein vollständiger Sensorabriss.
ECOFF	Der Offsetregler für das Cosinussignal hat seine Grenze erreicht. Ursache ist ein zu großer Signaloffset, ein ungültiger Wert zur Initialisierung des Reglers, ein teilweiser oder ein vollständiger Sensorabriss.
ESOFF	Der Offsetregler für das Sinussignal hat seine Grenze erreicht.

Zum Einstieg in die Analyse der Fehler-Flags bietet es sich an, sich über die Auswahl „ERRORS“ alle aufgetretenen Fehler-Flags anzeigen zu lassen. Hierbei werden alle Fehler-Flags über eine logische Oder-Funktion zu einem Auftrittereignis zusammengefasst und dargestellt. Welche Fehler-Flags tatsächlich aufgetreten sind, wird über ein Text-Label angezeigt:



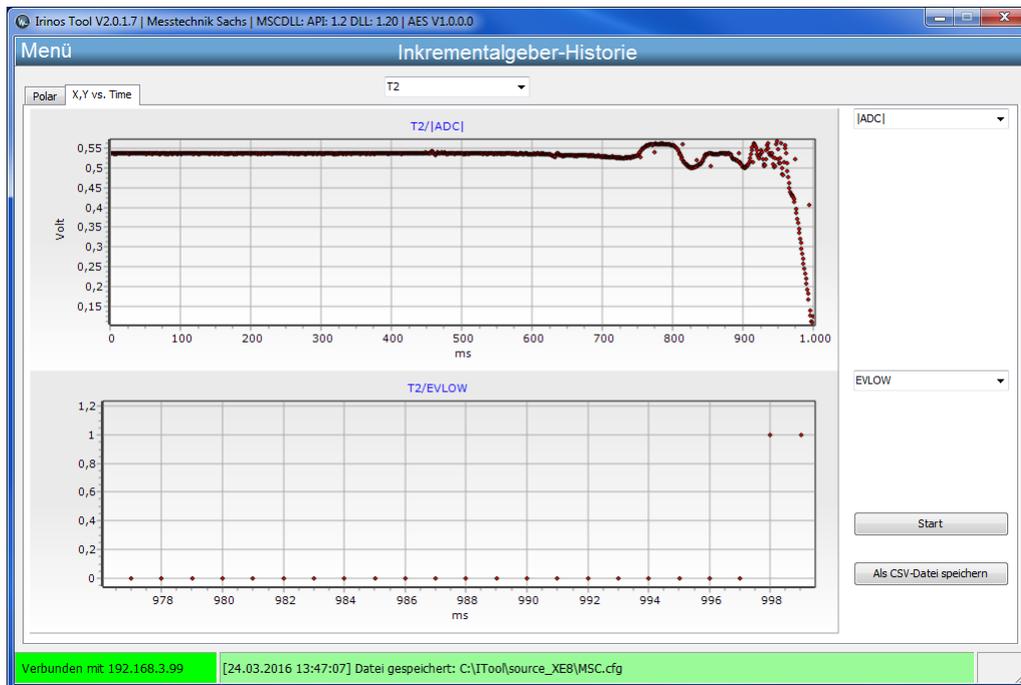
Betragsfunktion und ERRORS-Flag in der Zeitdarstellung

Ist bekannt welche Fehler-Flags aufgetreten sind, können diese einzeln im Auswahlménú angewählt werden (hier EVLOW) und zur zeitlichen Darstellung hinzugefügt werden:



Betragsfunktion und EVLOW-Flag in der Zeitdarstellung

Beide Tabellen sind mit einer Zoom-Funktion ausgestattet. Durch Ziehen von links-unten nach rechts-oben bei gedrückter linker Maustaste kann in die Darstellung hineingezoomt werden. Durch Ziehen von rechts-unten nach links-oben gelangt man zur ursprünglichen Darstellung zurück. Damit ist es möglich die wenig aussagekräftige Darstellung des Error-Flag EVLOW in eine detailliertere Darstellung zu überführen:



Betragsfunktion und EVLOW-Flag in der Zeitdarstellung (gezoomt)

In der höher aufgelösten Darstellung ist dann gut zu erkennen, dass das Error-Flag EVLOW erst bei den letzten beiden Sample-Punkten auftritt (Zeitwerte 998 und 999).

NmxDLL Referenz

4 NmxDLL Referenz

Geben Sie hier den Text ein.

4.1 Introduction

4.1.1 Imprint

Title	Nmx DLL reference manual
Provided by	Messtechnik Sachs GmbH Siechenfeldstraße 30/1 D-73614 Schorndorf Germany Phone +49 7181 26935-0 post@messtechnik-sachs.de
For use with	Irinos Measurement modules, Type IR and EC, Firmware version 2 and upwards
Copyright note	© 2019-2020 Messtechnik Sachs GmbH Messtechnik Sachs GmbH
Trademarks	All product names used in this manual are trademarks of their respective owners.
Change note	Subject to change without notice.
Release date	05.06.2020

4.1.2 Revision history

Version	Date	Changes
A	2019-05-05	First revision

B	201 9- 05- 24	Documentation of .Net-DLL ^[168] added.
C	202 0- 05- 08	New: High-Level sampling, type TFT ^[248] <i>Minimum DLL version: 1.1.0.11</i>
D	202 0- 06- 05	New: Position triggered sampling ^[245] <i>Minimum DLL version: 1.2.0.13</i> In addition, some minor typos have been corrected.

4.1.3 Legal notes

4.1.3.1 Terms of use for documentation & software

I. Protection rights and scope of use

Messtechnik Sachs provides operating instructions, manuals, documentation, and software programs - all collectively referred to as "LICENSED OBJECT" below - either on portable data storage devices (e.g. diskettes, CD ROMs, DVDs, etc.), in written (printed) form or in electronic form, for a fee and/or free of charge. The LICENSED OBJECT is subject to proprietary safeguarding provisions among other regulations. Messtechnik Sachs or third parties have protection rights for this LICENSED OBJECT. In so far as third parties have whole or partial right of access to this LICENSED OBJECT, Messtechnik Sachs has the appropriate rights of use. Messtechnik Sachs permits the user the use of the LICENSED OBJECT under the following conditions:

1.1) Scope of use for electronic documentation

- a) With the acquisition/purchase or relinquishment of a LICENSED OBJECT, you as the user acquire a simple, non-transferable right of use with regard to the respective LICENSED OBJECT. This right of use authorises the user to use the LICENSED OBJECT for the user's own, exclusively company-internal purposes on any number of machines within the user's business premises. This right of use includes exclusively the right to save the LICENSED OBJECT on the central processors (machines) used at the location.
- b) Irrespective of the form in which operating instructions and/or documentation are provided, the user may furthermore print out any number of copies on a printer at the user's location, providing this printout is printed with or kept in a safe place together with these complete terms and conditions of use and other user instructions.
- c) With the exception of the Messtechnik Sachs logo, the user has the right to use pictures and texts from the operating instructions/documentation for creating the user's own

machine and system documentation. The use of the Messtechnik Sachs logo requires written consent from Messtechnik Sachs. The user is responsible for ensuring that the pictures and texts used match the machine/system or the product.

d) Further uses are permitted within the following framework: Copying exclusively for use within the framework of machine and system documentation from electronic documents of all documented supplier components. Demonstrating to third parties exclusively under guarantee that no data material is stored wholly or partly in other networks or other data storage devices or can be reproduced there. Passing on printouts to third parties not covered by the regulation in item 3, as well as any processing or other use are not permitted.

1.2) Scope of use for software products

For any type of Messtechnik Sachs software including the associated documentation, the customer shall receive a non-exclusive, non-transferable and time-unlimited right of use on a certain hardware product or on a hardware product to be determined in individual cases. Messtechnik Sachs shall remain the owner of the copyright as well as of any other industrial property rights. The customer may make copies for back-up purposes only. Any copyright notes may not be removed.

2. Copyright note

Every LICENSED OBJECT contains a copyright note. In any duplication permitted under these provisions, the corresponding copyright note of the original document concerned must be included:

Example: © 2019, Messtechnik Sachs GmbH,
 D-73614 Schorndorf

3. Transferring the authorisation of use

The user can transfer the authorisation of use re. the respective LICENSED OBJECT as per these provisions in the scope and with the limitations of the conditions in accordance with items 1 and 2 completely to a third party. The third party must be made explicitly aware of these terms and conditions of use.

II. Exporting the LICENSED OBJECT

When exporting the LICENSED OBJECT or parts thereof, the user must observe the export regulations of the exporting country and those of the acquiring country.

III. Warranty

1. Messtechnik Sachs products are being further developed with regard to hardware and software. If the LICENSED OBJECT, in whatever form, is not supplied with the product, i.e. is not supplied on a data storage device as a delivery unit with the relevant product, Messtechnik Sachs does not guarantee that the

electronic documentation corresponds to every hardware and software status of the product. In this case, the printed documentation from Messtechnik Sachs accompanying the product is alone decisive for ensuring that the hardware and software status of the product matches that of the electronic documentation.

2. The information contained in an item of electronic documentation can be amended by Messtechnik Sachs without prior notice and does not commit Messtechnik Sachs in any way.

3. Messtechnik Sachs guarantees that the software program it created agrees with the change description and program specification but not that the functions included in the software run entirely without interruptions and errors or that the functions included in the software can run or meet the requirements in all combinations selected by and in all conditions of use designated by the acquirer.

IV. Liability/limitations on liability

1. Messtechnik Sachs provides LICENSED OBJECTS to allow the user to use - in conformity with the contract - Messtechnik Sachs products which require software for proper operation, or to assist the user in creating the user's machine and system documentation. In the case of electronic documentation which in the form of data storage devices does not accompany a product, i.e. which is not supplied together with that product, Messtechnik Sachs does not guarantee that the electronic documentation separately available/supplied matches the product actually used by the user.

The latter applies particularly to extracts of the documents for the user's own documentation. The guarantee and liability for separately available/supplied portable data storage devices, i.e. with the exception of electronic documentation provided on the Internet/Intranet, are limited exclusively to proper duplication of the software, whereby Messtechnik Sachs guarantees that in each case the relevant portable data storage device or software contains the latest status of the documentation. In respect of the electronic documentation on the Internet/Intranet, it is not guaranteed that this has the same version status as the last printed edition.

2. Furthermore, Messtechnik Sachs cannot be held liable for the lack of economic success or for damage or claims by third parties resulting from use of the LICENSED OBJECTS by the user, with the exception of claims arising from infringement of protection rights of third parties concerning the use of the LICENSED OBJECTS.

3. The limitations on liability as per paragraphs 1 and 2 do not apply if, in cases of intent or wanton negligence or lack of warranted quality, liability is absolutely necessary. In such a case, the liability of Messtechnik Sachs is limited to the damage recognisable by Messtechnik Sachs when the specific circumstances are made known.

V. Safety guidelines/documentation

Guarantee and liability claims in conformity with the regulations mentioned above (items III and IV) can only be made if the user has observed the safety guidelines of the documentation in conjunction with use of the machine and its safety guidelines or the terms and conditions of use of the software. The user is responsible for ensuring that the electronic documentation, which is not supplied with the product, matches the product actually used by the user.

4.1.3.2 Qualified personnel

The product system described in this documentation must only be handled by qualified personal according to the given scope of work. All documentation relevant for the scope of work must be observed, especially the safety and warning notes. Due to its education and experience, qualified personal is able to identify risks and possible dangers when using this products / systems.

4.1.3.3 Disclaimer

The content of this documentation has been carefully reviewed to comply with the documented hard- and software. We can, however, not exclude discrepancies and do therefore not accept any liability for the exact compliance. This documentation is reviewed regularly. Corrections may be contained in newer versions.

4.1.4 Preface

4.1.4.1 Purpose

This reference manual describes the functions of the NMX DLL for the use with the measurement system. The target audience is software developers, who want to integrate the DLL into their application software (measurement software).

4.1.4.2 Scope of this reference manual

This reference manual is valid for the industrial measurement system Irinos together with the NMX DLL. The NMX software interface is supported from Firmware Version 2 and upwards.

4.1.4.3 Required knowledge

Profound knowledge in PC based software development using Windows is required for integrating and using the NMX DLL.

4.1.4.4 Further documentation

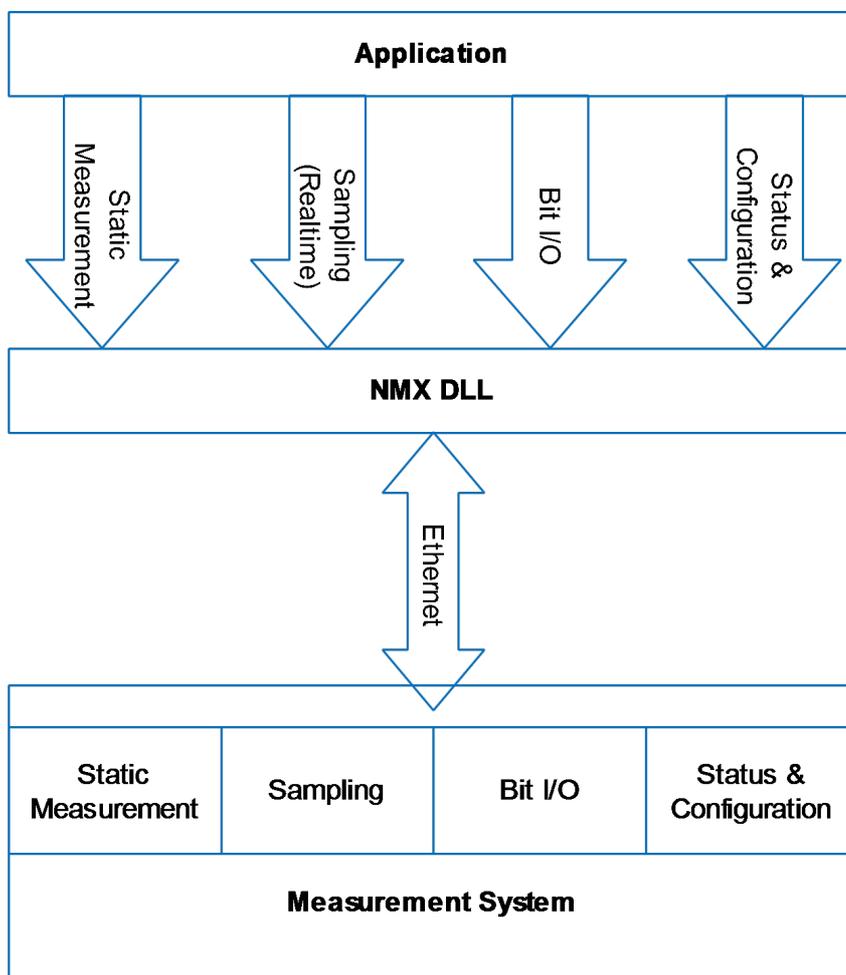
This reference manual is limited to the software interface of your measurement system.

For information about the measurement system, please consult its user manual and technical specification.

4.2 Nmx DLL Overview

The NMX DLL is the link between the application software (measurement software) and the Irinos-System. It supports the application in the following fields:

- Read measurement values from the measurement system. Reading [static or sampled](#)^[156] data is available.
- Read status information from the measurement system.
- Read / write digital in-/output data.
- Parametrize the measurement system.



The DLL provides a wide range of [function calls](#)^[169]. These allow for a flexible use of the measurement system.

For small systems with limited requirements, just a few of these functions are required.

In most use cases only one measurement system is connected. In this case only one connection is required.

For very special applications, multiple measurement systems can be connected. In this case up to 8 measurement systems can be connected in parallel. These are distinguished by their "Handle".

4.2.1 Static vs. Sampling

Two different types of getting measurement and I/O data are available, "static" and "sampling". Both can be used in parallel.

Static data acquisition is very easy to use and the perfect choice for many applications. It is always active. The typical use case are measurement applications, where the working piece is fixed ("static") during measurement.

The measurement and I/O data is updated continuously with an update rate of approximately 30 Hz. The data is neither synchronized nor transferred in realtime. The update rate is not guaranteed and can be longer due to data packet loss.

It is started automatically after connecting to a device. The NMX DLL then periodically request new data from the device. This data is copied into an internal buffer. It can be readout at any time.

Often the static measurement is also used to achieve an online-view of the current measurement values.

Sampling is used to gather synchronized real-time data from the device. The typical use case are dynamic measurement applications, where high speed and determinism are required, e.g. for getting measurement curves.

It can be limited in time or it can be endless. Once started, the measurement system (device) acquires the measurement and digital I/O data in realtime and stores it into the internal device buffer. From there it is transferred in packets to the NMX DLL without real-time requirements.

Before starting sampling, various parameters must be defined, like the measurements channels used, the digital in-/outputs used and the sample period. Depending on the amount of sampled data and the sample period, the sampling must be time limited or can be endless. Consult the user manual / data sheet of your measurement system to see the possibilities. Typically 1000 Samples/s are possible with a large number of measurement channels.

The NMX DLL supports two general types of sampling:

- **Low-Level sampling** offers the possibility to perform time-triggered measurements. It provides the highest flexibility.

- **High-Level sampling** provides a simplified interface for performing complex real-time measurement tasks, e.g. position-triggered sampling. Internally it always uses Low-Level sampling. Therefore it is always optional and the high-level routines could also be implemented into the application software directly.

4.2.2 Sampling Speed with Irinos

For all Irinos systems, the real-time capability is independent of the number of probes (measurement channels), since each Irinos box has its own measurement value buffer. With the

- **Irinos IR**, most boxes are able to acquire **20000 samples/s**, and with the
- **Irinos EC**, most boxes are able to acquire **4000 samples/s**.

Two quick rules of thumb for 99% of applications are:

- **If you limit the sampling speed to 1000 samples/s, there is no need to think about any details. Or:**
- **If you limit the sampling speed to 10000 samples/s and if you have a maximum of 4 Boxes, there is no need to think about any details.**
(With the Irinos EC, the max. speed still is 4000 samples/s.)

The sampling period used with the NMX DLL must be an integer multiple of the minimum sampling period of the system. **The following table lists typical sampling speeds:**

Sampling Speed [samples/s]	Sampling Period	Irinos IR	Irinos EC
20.000	50 µs	OK	not supported
10.000	100 µs	OK	not supported
6.666	150 µs	OK	not supported
5.000	200 µs	OK	not supported
4.000	500 µs	OK	OK

2.000	500 μ s	OK	OK
1.000	1000 μ s = 1 ms	OK	OK
500	2000 μ s = 2 ms	OK	OK
200	5000 μ s = 5 ms	OK	OK
100	10000 μ s = 10 ms	OK	OK

The memory is dimensioned so that the measured values can be **buffered for at least 10 seconds at the maximum sampling rate**. If the sampling rate is lower, this time increases accordingly.

Only the transmission time to the PC depends on the number of channels and the measuring rate. Since the data transmission to the PC and the PC itself do not have realtime capabilities, no guaranteed transmission time can be guaranteed. However, typically a transmission rate can be achieved, which is almost constant and thus close to realtime.

For time-limited real-time measurement, the transmission rate is relevant for calculating the typical time between "start of sampling" and "all measurement data available on the PC". This time is called "transfer time".

For endless measurement, the transmission rate is relevant for determining the maximum possible sampling speed.

Typically achievable transmission rates are listed in the following table:

IrinOS System	Typical transmission rate R_{TR-32} with 32 Bit measurement channels, e.g. Incremental probes	Typical transmission rate R_{TR-16} with 16 Bit measurement channels e.g. Inductive probes
----------------------	---	--

Irinos IR	<u>approx. 200.000</u> values/s	<u>approx. 400.000</u> values/s
Irinos EC	<u>approx. 80.000</u> values/s	<u>approx. 160.000</u> values/s

As shown in the table, the transmission rate also depends on the native data type(s) of the measurement channels used. If mixed types of measurement channels are used, then the typical transmission rate will be in between the given values.

Some examples are provided here to give a quick guidance. For a detailed examination, calculation formulas are given below.

Examples for Transfer Time for a time-limited sampling with a duration (Start -> Stop) of 10 seconds:

Configuration	Irinos IR	Irinos EC
32 Inductive Probes (16 Bit)	10000 samples/s: $10s + 0s = 10s$ -> Similar to realtime	4000 samples/s: $10s + 0s = 10s$ -> Similar to realtime
32 Incremental Probes (32 Bit)	10000 samples/s: $10s + 6s = 16s$ -> approximately 6 seconds after stop, the transfer is finished	4000 samples/s: $16s + 6s = 16s$ -> approximately 6 seconds after stop, the transfer is finished
32 Inductive Probes (16 Bit) + 4 Incremental Probes (32 Bit)	10000 samples/s: $10s + 0s = 10s$ -> Similar to realtime	4000 samples/s: $10s + 0s = 10s$ -> Similar to realtime

16 Inductive Probes (16 Bit) + 16 Incremental Probes (32 Bit)	10000 samples/s:	4000 samples/s:
	10s + 2s = 12s	10s + 2s = 12s
	-> approximately 2 seconds after stop, the transfer is finished	-> approximately 2 seconds after stop, the transfer is finished

Examples for maximum sampling speed for endless sampling:

Configuration	Irinos IR	Irinos EC
32 Inductive Probes (16 Bit)	Theoretical maximum: 12500 samples/s Next possible value: 10000 samples/s Recommended: ≤ 6666 samples/s	Theoretical maximum: 5000 samples/s Next possible value: 4000 samples/s Recommended: ≤ 2000 samples/s
32 Incremental Probes (32 Bit)	Theoretical maximum: 6250 samples/s Next possible value: 5000 samples/s Recommended: ≤ 4000 samples/s	Theoretical maximum: 2500 samples/s Next possible value: 2000 samples/s Recommended: ≤ 1000 samples/s
32 Inductive Probes (16 Bit) + 4 Incremental Probes (32 Bit)	Theoretical maximum: 10000 samples/s Next possible value: 10000 samples/s Recommended: ≤ 6666 samples/s	Theoretical maximum: 4000 samples/s Next possible value: 4000 samples/s Recommended: ≤ 2000 samples/s

16 Inductive Probes (16 Bit) + 16 Incremental Probes (32 Bit)	Theoretical maximum: 8333 samples/s Next possible value: 6666 samples/s Recommended: \leq 5000 samples/s	Theoretical maximum: 3333 samples/s Next possible value: 2000 samples/s Recommended: \leq 2000 samples/s
64 Inductive Probes (16 Bit)	Theoretical maximum: 6250 samples/s Next possible value: 5000 samples/s Recommended: \leq 4000 samples/s	Theoretical maximum: 2500 samples/s Next possible value: 2000 samples/s Recommended: \leq 1000 samples/s

Formulas

Following a few formulas are provided to get estimations. These formulas use the following variables:

Variable	Meaning	Unit
t_{Transfer}	Transfer Time: Time between "Start of sampling" and "All measurement data available on the PC"	s -> seconds
t_{Sampling}	Sampling Time: Time between "Start" and "Stop" of sampling	s -> seconds

VS_{Max}	Maximum sampling speed for endless sampling	values / s / channel --> same as: samples / s
R_{TR}	Typical Transmission rate, see table above.	values / s
N_{MCH}	Number of measurement channels used	channels
N_{MCH-16}	Number of 16 bit measurement channels used	channels
N_{MCH-32}	Number of 32 bit measurement channels used	channels
N_{Samples}	Total number of samples to be recorded	values / channel

Each formula is provided in a simplified form and in a detailed form. The simplified form is sufficient for most applications as a quick check. The detailed form is especially used, when very high performance is required.

Required value	Simplified	Detailed
<p>Transfer time for time-limited sampling</p> <p>(Time between "start of sampling" and "all measurement data is available at the PC")</p>	<p>Formula 1:</p> $t_{Transfer} = \frac{N_{Samples} * N_{MCH}}{R_{TR-32}}$ <p>Note: $t_{Transfer}$ is always $\geq t_{Sampling}$</p>	<p>Formula 2:</p> $t_{Transfer} = \frac{N_{Samples} * (N_{MCH-16} + 2 * N_{MCH-32})}{R_{TR-16}}$ <p>Note: $t_{Transfer}$ is always $\geq t_{Sampling}$</p>

Max. sampling speed for endless sampling	Formula 3: $vS_{Max} = \frac{R_{TR-32}}{N_{MCH}}$	Formula 4: $vS_{Max} = \frac{R_{TR-16}}{N_{MCH-16} + 2 * N_{MCH-32}}$
---	---	---

Example for Formula 1:

In the measurement application with the Irinos IR, 21 inductive probes, 4 analogue channels and 3 incremental probes/encoders are used. The measurement has a duration of 5 seconds at 10000 samples/s.

$$N_{MCH} = 21 + 4 + 3 = 28 \text{ channels}$$

$$NSamples = 5s * 10000 \text{ values/s/channel} = 50000 \text{ values/channel.}$$

$$R_{TR-32} = 200000 \text{ values/s}$$

$$t_{Transfer} = \frac{50000 \text{ values/channel} * 28 \text{ channels}}{200000 \text{ values/s}} = 7s$$

--> 2 seconds after stop of sampling, all data is available.

Example for Formula 2:

In the measurement application with the Irinos IR, 21 inductive probes, 4 analogue channels and 3 incremental probes/encoders are used. The measurement has a duration of 5 seconds at 10000 samples/s.

$$N_{MCH-16} = 21 + 4 = 25 \text{ channels}$$

$$N_{MCH-32} = 3 \text{ channels}$$

$$NSamples = 5s * 10000 \text{ values/s/channel} = 50000 \text{ values/channel.}$$

$$R_{TR-16} = 400000 \text{ values/s}$$

$$t_{Transfer} = \frac{50000 \text{ values/channel} * (25 \text{ channels} + 2 * 3 \text{ channels})}{400000 \text{ values/s}} = 3,875s$$

--> Since $t_{Transfer} < t_{Sampling}$ all data is available immediately after stop of sampling.

Example for Formula 3:

In the measurement application with the Irinos EC, 11 inductive probes + 1 incremental encoder are used.

$$N_{\text{MCH}} = 11 + 1 = 12 \text{ channels}$$

$$R_{\text{TR-32}} = 80000 \text{ values/s}$$

$$vS_{\text{Max}} = \frac{80000 \text{ values/s}}{12 \text{ channels}} = 6666 \frac{\text{values}}{\text{s}} / \text{channel}$$

--> The maximum speed of the Irinos EC, which is 4000 samples/s, can be used.

Example for Formula 4:

In the measurement application with the Irinos IR, 31 inductive probes + 6 incremental encoders are used.

$$N_{\text{MCH-16}} = 31 \text{ channels}$$

$$N_{\text{MCH-32}} = 6 \text{ channels}$$

$$vS_{\text{Max}} = \frac{400000 \text{ values/s}}{31 \text{ channels} + 2 * 6 \text{ channels}} = 9302 \frac{\text{values}}{\text{s}} / \text{channel}$$

--> 5000 samples/s can be used as maximum sample rate.

4.2.3 Data Types

The NMX DLL provides all

- measurement values as signed 32 bit values ("signed long") and all
- digital in-/outputs as unsigned 8 bit values ("unsigned char").

The native format of the measurement values can be different. For inductive probes the native format is for example signed 16 bit ("signed short"). To simplify the DLL interface, this native format is converted / "casted" to the common 32 Bit format within the DLL.

Example: An inductive probe has the value -9152, which is 0xDC40. This 16 Bit value is then converted to the 32 Bit value 0xFFFFDC40, which is also -9152.

If required, the native data type of a measurement channel can be retrieved from the NMX DLL using the function [NMX_GetChannelInfo_1](#)^[206]. However, in most cases the conversion factor to a physical unit is of interest and not the data type.

Digital in-/outputs are always transferred byte-wise, which means that each 8 in-/outputs are represented by 1 byte.

Example: A system has 32 digital inputs. These are represented by $32 / 8 = 4$ input bytes, whereas

- Byte 0 contains the inputs 1..8
- Byte 1 contains the inputs 9..16
- etc.

4.2.4 Technical Background

The NMX DLL uses the Windows API functions for IP based communication, thread management and timing.

Inside the NMX DLL separate threads are running, which control the communication and the [notification](#)¹⁹². The NMX DLL functions provide data to these threads and vice versa.

Some of the threads use normal thread priority while others use highest thread priority.

Please note: Even though the NMX DLL is multi-threaded, its function calls are not thread safe against each other. This means that all DLL functions must be called from within the same thread!

Communication to the measurement system is based on UDP/IPV4. The DLL automatically retransmits a data packet, if it has been lost. A direct ethernet connection between the measurement system and the PC is advised. Complex network structures, e.g. routing, tunneling, VPN, etc. are not supported due to timing efficiency. If you use these, you do it on your own risk.

The NMX DLL has been designed in C++ (VS2017) and uses the "**stdcall**" **calling convention**. A C based header file is provided.

Example applications for various programming environments are provided. It is good practice to start with one of these.

4.2.5 Limitations

The NMX DLL has several limitations. These have been selected such that in almost all cases they are of theoretical nature.

Note that your measurement system may have limitations below those of the NMX DLL.

The NMX DLL limitations are:

- Max. number of measurement boxes: 64
- Max. number of measurement channels: 256
- Max. number of sampling elements: 512
- Max. number of handles (-> simultaneous connections): 8

4.2.6 Hardware Requirements

The NMX DLL has no special hardware requirements.

Practical tests have shown that the **CPU load is almost negligible** and all of today's CPUs are sufficient.

To give an example: using the Visual C++ demo application, a 2 weeks sampling at 1kSample/s using 64 measurement channels consumed a total CPU time of 22 seconds (CPU i5-6300U). Usually the CPU performance required for the measurement application is far higher than for the DLL.

The **memory usage** mainly depends on the buffer, which is required for sampling. This means, it is defined by you (see [NMX Sampling PrepareTime 1](#)²³⁴). **For typical applications, the memory consumption is very low (< 10 MBytes).**

4.2.7 Versions

The version number of the NMX DLL consists of 4 parts, which are separated by a dot, e.g. V1.3.0.12. The meaning of the parts is:

Part of the version number	Meaning
1. Part, in the example: 1	"Major" version number It is incremented, if the NMX DLL is completely redesigned (-> happens seldom).
2. Part, in the example: 3	"Minor" version number It is incremented, if new functionality has been implemented.
3. Part, in the example: 0	"Patch" It is incremented, if one or more bugs have been fixed.
4. Part, in the example: 12	"Build" Internal number for revision identification.

To ensure a long-term backward compatibility, newer versions may have additional functionality, but existing functionality remains unchanged (except bug-fixes). Therefore each function call has a separate suffix at its end: `_1`

An example is `NMX_StaticGet32_1`.

If a newer version of an existing function is implemented, a copy of the existing function call is made and the suffix is incremented. Following the example, a newer version would be named `NMX_StaticGet32_2`, whereas the existing function call remains unchanged.

It is strongly recommended to check for a minimum DLL version after startup of your measurement software (especially the Major and Minor part). Use [NMX_GetDllVersion_1](#)¹⁸⁰ for this purpose.

The NMX DLL may support functionality, which is not supported by the measurement system ("device"). This can be either since it is limited in functionality or since its firmware version is outdated. The function call will then return with the [return code](#)^[176] "NST_REQ_VERS_NOSUPPORT".

4.2.8 INI-File

Via an INI-File NmxDLL.ini, the behaviour of the NMX DLL can be modified. Currently the following modifications are possible:

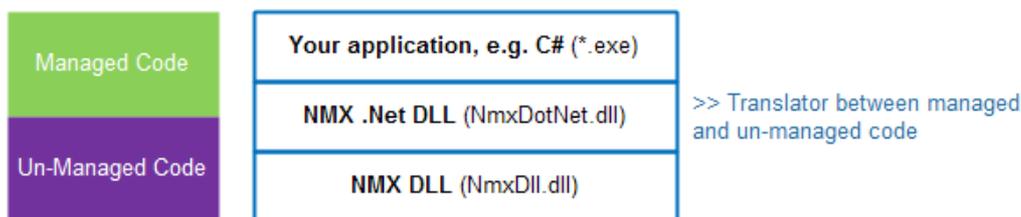
- Enable writing a log file NmxDLL.log.
- Change timing settings.

This INI-File should only be used if advised by the support.

4.2.9 .NET Wrapper DLL

For .Net based environments, like for example C# or VB.Net, a wrapper DLL is available (NmxDotNet.dll).

This DLL provides managed .Net function calls for your application. Internally, it converts between the managed .Net-World and the unmanaged native code of the NMX DLL:



In addition, the .Net DLL is much more convenient to use. There is for example no need to declare any function prototypes, since these are already embedded into the DLL.

Please note:

- The wrapper DLL uses the standard NMX DLL. Therefore both DLLs (NmxDLL.dll and NmxDotNet.dll) must be placed in the same folder.
- Since the wrapper DLL calls native code, it is "unsafe code" from the .Net perspective.
By default, unsafe code is not allowed in .Net applications. This setting

must be changed in the "Build" section of your project properties, to allow unsafe code.

- The API of the wrapper DLL is slightly different to the native DLLs API. For example arrays are handled differently. However, the basic concept is the same.

4.3 API (programming interface)

4.3.1 Function calls overview

The following table provides a list of all function calls supported by the NMX DLL.

Don't be frightened by the long list of functions. Small measurement applications may need just a few of these function calls. See the HowTo "[Small Measurement Application](#)"²⁵⁵ in this manual.

There is quite a simple rule: the more sophisticated your measurement application is, the more function calls you will need.

Category	Function call	Description	Min. DLL version
Miscellaneous	NMX_GetDllVersion_1 ¹⁸⁰	Get version of the NMX DLL.	V1.0.0.0
	NMX_SystemReset_1 ¹⁸¹	Reset the device via Software.	V1.0.0.0
Connecting / Disconnecting	NMX_DeviceIPv4Open_1 ¹⁸⁹	Establish a connection to the device via IPV4.	V1.0.0.0
	NMX_DeviceClose_1 ¹⁹¹	Close a connection to a device.	V1.0.0.0

Category	Function call	Description	Min. DLL version
Notifications	NMX_RegisterMessage 1 	Register a notification message.	V1.0.0.0
	NMX_RegisterCallback 1 	Register a notification callback.	V1.0.0.0
Get device information	NMX_GetBoxCount 1 	Get number of measurement boxes available.	V1.0.0.0
	NMX_GetBoxInfo 1 	Get information about a measurement box (digital type plate).	V1.0.0.0
	NMX_UpdateChannelInfo 1 	Re-Read measurement channel information.	V1.0.0.0
	NMX_GetChannelCount 1 	Get number	V1.0.0.0

Category	Function call	Description	Min. DLL version
		of measurement channels available.	
	NMX_GetChannelInfo_1 ^[206]	Get information about measurement channel (digital type plate).	V1.0.0.0
	NMX_GetDigitalInputInfo_1 ^[211]	Get information about digital input byte.	V1.0.0.0
	NMX_GetDigitalOutputInfo_1 ^[213]	Get information about digital output byte.	V1.0.0.0
Static measurement (Non-Realtime)	NMX_StaticGet32_1 ^[214]	Read <ul style="list-style-type: none"> • static measurement values, • digital input data, • hardware 	V1.0.0.0

Category	Function call	Description	Min. DLL version
		status and • current box events.	
	NMX_StaticSetMedianDepth 1 ^[220]	Set median filter for static measurement values.	V1.0.0.0
	NMX_SetOutputs 1 ^[221]	Set digital outputs.	V1.0.0.0
	NMX_DisableOutputUpdate 1 ^[222]	Disable updating digital outputs.	V1.0.0.0
	NMX_DigitalIoConfig 1 ^[223]	Configure digital I/O.	V1.0.0.0
	NMX_DigitalOutputsGetState 1 ^[224]	Get current state of digital outputs.	V1.0.0.0
Sampling LowLevel (Time-Trigged Realtime)	NMX_Sampling_GetMaxSpeed 1 ^[225]	Get maximum sampling speed.	V1.0.0.0

Category	Function call	Description	Min. DLL version
Measurement)	NMX Sampling Reset 1 	Reset sampling .	V1.0.0.0
	NMX Sampling AddChannelsAll 1 	Add all measurement channels to the list of sampling elements .	V1.0.0.0
	NMX Sampling AddChannel 1 	Add a single measurement channel to the list of sampling elements .	V1.0.0.0
	NMX Sampling AddDigiInAll 1 	Add all digital input bytes to the list of sampling elements .	V1.0.0.0
	NMX Sampling AddDigiInByte 1 	Add a single digital input byte to the list of sampling	V1.0.0.0

Category	Function call	Description	Min. DLL version
		elements .	
	NMX Sampling AddDigiOutAll 1 ^[232]	Add all digital output bytes to the list of sampling elements .	V1.0.0.0
	NMX Sampling AddDigiOutByte 1 ^[233]	Add a single digital output byte to the list of sampling elements .	V1.0.0.0
	NMX Sampling PrepareTime 1 ^[234]	Prepare sampling .	V1.0.0.0
	NMX Sampling Start 1 ^[236]	Start sampling .	V1.0.0.0
	NMX Sampling Stop 1 ^[237]	Stop sampling .	V1.0.0.0
	NMX Sampling ReadColumn32 1 ^[237]	Read sampled data "column-wise".	V1.0.0.0
	NMX Sampling ReadRow32 1 ^[240]	Read sampled	V1.0.0.0

Category	Function call	Description	Min. DLL version
		data "row-wise".	
	NMX_Sampling_GetStatus_1 ^[242]	Get current sampling status.	V1.0.0.0
Sampling HighLevel (Application specific Realtime Measurement)	NMX_Sampling_PrepareCustomTFT_1 ^[248]	Prepare application specific sampling of the type "Trigger + Filter + Tail".	V1.1.0.11
Diagnostics	NMX_DiagClearEvent_1 ^[251]	Clear event at Box.	V1.0.0.0
	NMX_DiagGetEventText_1 ^[252]	Get text describing the event.	V1.0.0.0
	NMX_SetDateTime_1 ^[254]	Set current date & time.	V1.0.0.0

4.3.2 Function Return Codes (NMX_STATUS)

The function calls of the NMX DLL almost all have the same return value of the type NMX_STATUS. Most of the return values will rarely occur. The most common ones are shown in bold in the following table.

Note for the [.Net DLL](#)¹⁶⁸: In the .Net DLL, the return values are represented by the enum type NMX_MSTATUS instead by a binary value. Therefore NST_SUCCESS is for example NMX_MSTATUS.SUCCESS.

The return values are defined as follows:

Return value	Hex representation	Description	Possible reasons (excerpt)
NST_SUCCESS	0x00000000	Everything OK.	
NST_HANDLE_INVALID	0xF0000000	Invalid handle.	<ul style="list-style-type: none"> • Connection has not been established yet. • Connection has already been closed.
NST_HANDLE_TOO_MANY	0xF0000001	Too many handles exists.	Too many connections have been opened. The limit is 8.
NST_CONNECT_OPEN_FAILED	0xF0000010	Failed connecting to a device.	<ul style="list-style-type: none"> • Device is not connected to the PC. • Network settings (IP address & Port numbers) are invalid.
NST_NOT_CONNECTED	0xF0000011	No connection established.	
NST_NOT_AVAILABLE	0xF000001F	The requested data is not available.	No connection established or invalid data received from the device.
NST_SEND_SIZE_TOO_LARGE	0xF0000020	Too much data to send.	
NST_DX_TIMEOUT_1	0xF0000021		<ul style="list-style-type: none"> • The network connection has been interrupted shortly or permanently.
NST_DX_TIMEOUT_2	0xF0000022	A data exchange	<ul style="list-style-type: none"> • PC timing is too slow, e.g.

4.3.3 Connection Handle

The NMX DLL allows having multiple connections to different devices, even though in most applications one connection to one device is used.

In order to distinguish between multiple connections, a unique handle is assigned for each of them by the NMX DLL.

Technically spoken, a handle is nothing else than a pointer to an address space managed by the NMX DLL. This is a common programming technique.

The pointer itself resides outside the NMX DLL in the user application. It is assigned if a connection has been established successfully (see [NMX_DeviceIPv4Open_1](#)^[189]). It is then used in almost every DLL function, to identify the relevant connection. It is deleted by closing the connection (see [NMX_DeviceClose_1](#)^[191]).

The Handle has the following type definition (C-Code):

```
typedef void* NMX_PHANDLE;
```

For each connection a handle must be declared. In case only one connection is used, the declaration looks as follows:

```
NMX_PHANDLE pHandle = NULL;
```

Note for the [.Net DLL](#)^[168]: Using the .Net DLL, the handle uses the type `System::IntPtr`. The basic concept is the same. In case only one connection is used, the declaration in C# looks as follows:

```
IntPtr pDevice = IntPtr.Zero;
```

Further information, if you use multiple connections / handles:

Inside the NMX DLL, each connection / handle has its own memory and its own communication tasks. Talking in object orientated programming, a separate object is created for each handle.

If [notifications](#)^[192] are used, these must be registered for each handle separately.

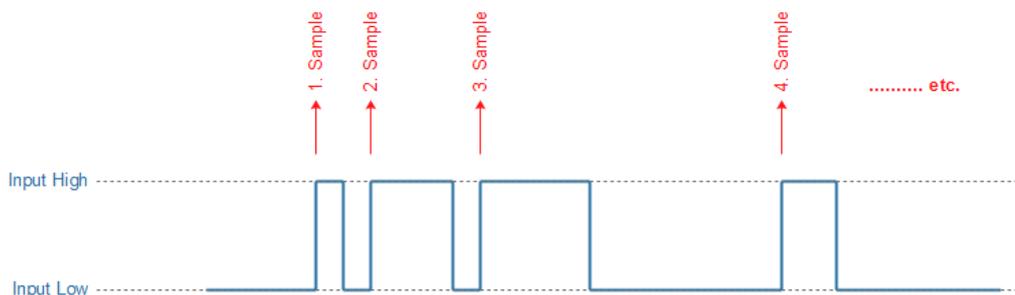
4.3.4 Trigger Modes

Certain sampling functions allow using a digital input for triggering. The following Trigger-Modes are implemented.

Please note that not all trigger modes are supported by every function.

Trigger Mode 1 = "Edge"

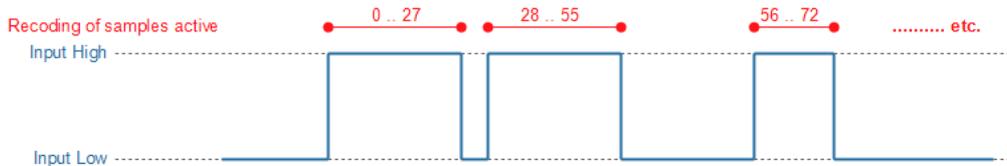
Using the edge trigger, exactly one sample is stored every time an edge of the digital input occurs. The following example shows this for "rising edge":



Please note that the maximum input frequency of the digital input must be \leq sampling frequency. Otherwise samples could be lost.

Trigger Mode 2 = "Level"

Using the level trigger, measurement values are sampled while the digital input is high (for polarity "high") or low (for polarity "low"). The following example shows this for polarity "high":



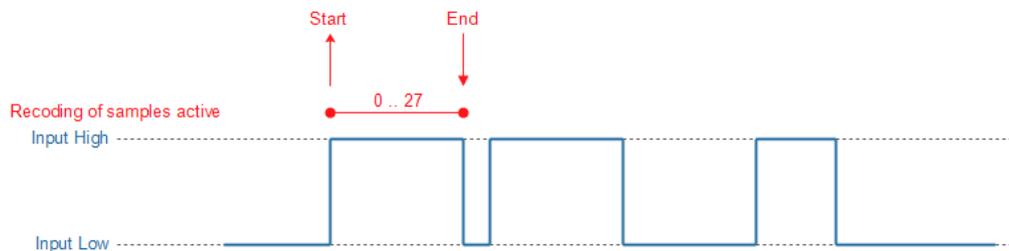
Trigger Mode 3 = "Edge Start"

Using the trigger "Edge Start", recording of sampled data is stored after the first edge of the digital input. Any further edges of the digital input have no effect. The following example shows this for "rising edge":



Trigger Mode 4 = "Level Once"

The trigger mode "Level Once" is similar to the "Level Trigger" (Mode 2), except that sampling is stopped automatically once the trigger condition becomes inactive. The following example shows this for polarity "high":



4.3.5 Miscellaneous

4.3.5.1 NMX_GetDllVersion_1

This function returns the current version of the NMX DLL.

Definition

```
void NMX_GetDllVersion_1(
    unsigned short* pusMajor,
    unsigned short* pusMinor,
    unsigned short* pusPatch,
    unsigned short* pusBuild);
```

Parameter

pusMajor

Major version of the NMX DLL.

pusMinor

Minor version of the NMX DLL.

pusPatch

Patch version of the NMX DLL.

pusBuild

Build number of the NMX DLL.

Typical function call (C example)

```
NMX_GetDllVersion_1(&usMajor, &usMinor, &usPatch, &usBuild);
```

[.Net DLL](#)^[168] specific implementation

```
System::Void GetDllVersion_1(System::UInt16 %pusMajor,  
System::UInt16 %pusMinor, System::UInt16 %pusPatch,  
System::UInt16 %pusBuild);
```

Comments

See chapter "[Versions](#)^[166]" for more information.

4.3.5.2 NMX_SystemReset_1

This function allows restarting the whole measurement system (device). Use this function only if advised by the support.

Definition

```
NMX_STATUS NMX_SystemReset_1(  
    NMX_PHANDLE pHandle,  
    unsigned long ulDelayMaster,  
    unsigned long ulDelaySlave);
```

Parameter

pHandle

[Connection Handle](#)¹⁷⁸

ulDelayMaster

Time in ms, until the Master-Box (-> first measurement box) will be restarted.

ulDelaySlave

Time in ms, until all Slave-Boxes will be restarted.

Typical function call (C example)

```
NMX_SystemReset_1(pHandle, 2000, 500);
```

[.Net DLL](#)¹⁶⁸ specific implementation

```
NMX_MSTATUS SystemReset_1(System::IntPtr pHandle,  
System::UInt32 ulDelayMaster, System::UInt32 ulDelaySlave);
```

Comments

The delay for the Master must be longer than the delay for the Slaves. Use the following values:

```
ulDelayMaster = 2000;
```

```
ulDelaySlave = 500;
```

The reset command is send to all Slave-Boxes via the Link interface between the boxes. If the Link interface does not work properly, no slave can be reset.

4.3.5.3 NMX_ChannelSetParameter_1

This function allows changing a channels parameters during operation. The parameters depend on the measurement channel type. It is used for selected measurement channels, e.g. for incremental encoders.

Definition

```
NMX_STATUS NMX_ChannelSetParameter_1(  
    NMX_PHANDLE pHandle,
```

```
    unsigned long ulChannelNo,  
    char* pcStringTx, unsigned long ulSizeofTxString,  
    char* pcStringRx, unsigned long ulMaxSizeofRxString,  
    unsigned long* pulSizeofRxString);
```

Parameter

pHandle

[Connection Handle](#) 

ulChannelNo

Number of the measurement channel.
The first channel has the number 0.
Having a device with 8 measurement channels, these are numbered 0..7.

pcStringTx

ASCII-encoded string, which shall be send to the device.

ulSizeofTxString

Size of pcStringTx in characters/bytes.

pcStringRx

Buffer for the response-string, which is received from the device.

ulMaxSizeofRxString

Size of the buffer pcStringRx in characters/bytes.

pulSizeofRxString

Length of the response-string, which has been received from the devices.

Typical function call (C example)

```
char acStrTx[] = "#0;REFOFF#";  
NMX_ChannelSetParameter_1(pHandleNmx, 2, acStrTx,  
strlen(acStrTx), acStrRx, sizeof(acStrRx), &ulRxSize);
```

Strings, which can be sent to incremental en-/decoder channels

`#{Position};{Reference index on/off}#`

Position

- New position value for this measurement channel. This allows setting the position of the measurement channel.
- * if the position of the measurement channel shall not be changed. This is required, if only the reference index shall be turned on or off.
- ~ in order to reset the gain- and offset-control of the measurement channel. The position value will be reset to 0. This makes only sense for 1Vpp channels.

Reference index on/off

Permissible parameter values:

- REFON to enable the reference index.
In addition, the statusbit "Refmark" (see [HardwareStatus^{\[214\]}](#)) will be enabled now, if the reference index is crossed.
- REFOFF to disable the reference index.

If the reference index is enabled, the position of the measurement channel will be set to 0, if the index is crossed.

Response string from the measurement system / measurement channel for incremental encoder

```
#0# Success
#-2# Position parameter in string invalid
#-3# Position parameter in string invalid
#-99# General syntax error of the request string
```

Response string from the Irinos-System / measurement channel does not support this opcode

#-98#

Examples for request strings to incremental measurement channels

#-2000;REFOFF#

The current position of the selected measurement channel will be set to -2000. The reference index will be disabled.

#*;REFON#

The reference index of the selected measurement channel will be enabled. The position will not be changed.

#~;REFOFF#

The gain- and offset control of the selected measurement channel will be reset. The position will be set to 0. The reference index will be disabled.

[.Net DLL](#)¹⁶⁸ specific implementation

```
NMX_MSTATUS ChannelSetParameter_1(System::IntPtr pHandle,  
System::UInt32 ulChannelNo, System::String ^%strExchange);
```

- *Since strExchange is a managed string, no additional string length information must be provided in the function call.*
- *The string strExchange is send to the device.
On success, the received string will be returned via the same parameter strExchange.*
- *Make sure to use only ASCII-characters. The conversion from unicode to ASCII is made within the wrapper DLL.*

Comments for incremental measurement channels

The error flags and the status bit "Refmark" will be cleared (these can be readout via the [hardware status](#)^[214]). Exception: This does not apply, if the character * is used for the parameter "position".

Notes for 1Vpp measurement channels:

If an error is detected at the 1Vpp-inputs, the signal levels of the incremental encoder are or have been out of specification. After an error has occurred, it is recommended to reset the gain- and offset-control by using the character ~ as the position parameter.

After resetting the gain- and offset-control, it takes a few signal periods until the control has determined the optimal parameters. During this process, the interpolation accuracy is limited. The measurement values can be inaccurate (however, no increments are lost). Further, the signal tolerance is limited during this process.

4.3.5.4 NMX_ChannelSetConfig_1

This function allows changing a channels parameters during operation. The parameters depend on the measurement channel type. It is used for selected measurement channels, e.g. for incremental encoders.

Definition

```
NMX_STATUS NMX_ChannelSetConfig_1(  
    NMX_PHANDLE pHandle,  
    unsigned long ulChannelNo,  
    char* pcStringTx, unsigned long ulSizeofTxString,  
    char* pcStringRx, unsigned long ulMaxSizeofRxString,  
    unsigned long* pulSizeofRxString);
```

Parameter

pHandle

[Connection Handle](#)^[178]

ulChannelNo

Number of the measurement channel.

The first channel has the number 0.

Having a device with 8 measurement channels, these are numbered 0..7.

pcStringTx

ASCII-encoded string, which shall be send to the device.

ulSizeofTxString

Size of pcStringTx in characters/bytes.

pcStringRx

Buffer for the response-string, which is received from the device.

ulMaxSizeofRxString

Size of the buffer pcStringRx in characters/bytes.

pulSizeofRxString

Length of the response-string, which has been received from the devices.

Typical function call (C example)

```
char acStrTx[] = "#1VSS;1#";  
NMX_ChannelSetConfig_1(pHandleNmx, 2, acStrTx, strlen(acStrTx),  
acStrRx, sizeof(acStrRx), &ulRxSize);
```

Strings, which can be sent to incremental en-/decoder channels

```
#{Configuration type};{Store}#
```

Configuration type

„1VSS“ to change an incremental channel type to 1Vpp.

„TTL“ to change an incremental channel type to TTL / RS422 with 1x-decoding (1 increment = 1 digit).

„TTL4X“ to change an incremental channel type to TTL / RS422 with 4x-decoding (1 increment = 4 digits).

Store

„0“: The change remains active until the next system restart.
Afterwards the old configuration becomes active again.

„1“: The change applies permanently.

Response string from the measurement system / measurement channel for incremental encoder

#0# Success

#-2# Position parameter in string invalid

#-3# Position parameter in string invalid

#-99# General syntax error of the request string

Response string from the Irinos-System / measurement channel does not support this opcode

#-98#

[.Net DLL](#)¹⁶⁸ specific implementation

```

NMX_MSTATUS ChannelSetConfig_1(System::IntPtr pHandle,
System::UInt32 ulChannelNo, System::String ^%strExchange);

```

- Since *strExchange* is a managed string, no additional string length information must be provided in the function call.
- The string *strExchange* is send to the device.
On success, the received string will be returned via the same parameter *strExchange*.
- Make sure to use only ASCII-characters. The conversion from unicode to ASCII is made within the wrapper DLL.

Comments for incremental measurement channels

After configuration, the position of the incremental input channel is reset.

By reconfiguring an incremental input channel after start of the application software, it does not matter how a measurement channel is pre-configured. This allows a quick replacement of a measurement Box without manual reconfiguration of the "new" Box.

4.3.6 Connecting / Disconnecting

A connection to the device must be established before any other function will work properly (exception: [NMX_GetDllVersion_1](#)^[180]).

4.3.6.1 NMX_DeviceIPv4Open_1

This function is used to establish a connection to a device (measurement system) via IPV4.

See also the [HowTo](#)^[255]-Guide.

Definition

```
NMX_STATUS NMX_DeviceIPv4Open_1(  
    unsigned char ucIp3,  
    unsigned char ucIp2,  
    unsigned char ucIp1,  
    unsigned char ucIp0,  
    unsigned short usPortCmd,  
    unsigned short usPortData,  
    NMX_PHANDLE *ppHandle);
```

Parameter

ucIp3, ucIp2, ucIp1, ucIp0

IP-Address of the device. The factory default address is 192.168.3.99.

For using this address, the parameters are as follows:

```
ucIp3 = 192;  
ucIp2 = 168;  
ucIp1 = 3;  
ucIp0 = 99;
```

usPortCmd

Port number for the DLLs communication channel, which is used for exchanging commands. Use 22517.

usPortData

Port number for the DLLs communication channel, which is used for exchanging data. Use 22516.

ppHandle

If the connection has been established successfully, a connection handle will be returned via this pointer. Otherwise the handle remains unchanged.

Typical function call (C example)

```
NMX_DeviceIPv4Open_1(192, 168, 3, 99, 22517, 22516, &pHandle);
```

[.Net DLL](#)^[188] specific implementation

```
NMX_MSTATUS DeviceIPv4Open_1(  
    System::Byte ucIp3,  
    System::Byte ucIp2,  
    System::Byte ucIp1,  
    System::Byte ucIp0,  
    System::UInt16 usPortCmd,  
    System::UInt16 usPortData,  
    System::IntPtr %ppHandle);
```

Comments

- NMX_PHANDLE is a pointer to a handle (type void*). It is required for all subsequent connection calls.
- It is strongly recommended to read the IP-Address and the Port-Numbers from an INI-File, XML-File, Registry or something similar. Do not hard code these.
- If the connection is not established after having integrated this function into your application, first check the network connection to the device (e.g. by calling its WebServer or via ping).
- Before closing your application, always ensure that this connection is closed before via [NMX_DeviceClose_1](#)^[191].

- Multiple connections can be established to different devices. In this case a separate handle is provided for each connection. However, this is very rarely required. Please read also the chapter about [limitations](#)^[165].

4.3.6.2 NMX_DeviceClose_1

This function is used to close a connection to a device (measurement system).

See also the [HowTo](#)^[257]-Guide.

Definition

```
NMX_STATUS NMX_DeviceClose_1(  
    NMX_PHANDLE *ppHandle);
```

Parameter

ppHandle

Pointer to [Connection Handle](#)^[178].

Typical function call (C example)

```
NMX_DeviceClose_1(&ppHandle);
```

[.Net DLL](#)^[168] specific implementation

```
NMX_MSTATUS DeviceClose_1(System::IntPtr %ppHandle);
```

Comments

If the connection is not established, nothing happens except that NST_HANDLE_INVALID is returned.

An established connection must be closed before terminating your application. It is good practice calling this function in any case before terminating.

4.3.7 Notifications

Using notifications, the application can be informed about different events. The use of notifications is recommended, but not required.

The following notifications are available:

Category	Notification	Hex Value	Description
Connecting / Disconnecting	NMXNOTIFY_DISCONNECT	0x000	The connection to the device (measurement system) has been closed. Prior to this event the function NMX_DeviceClose_1 ¹⁹¹ has been called.
	NMXNOTIFY_FAILURE	0x000	Permanent failure during data exchange (e.g. timeout). A typical reason is a broken network connection.
	NMXNOTIFY_RECONNECT	0x000	The connection has been re-established automatically. Prior to this event, the event NMXNOTIFY_FAILURE_DATA_EXCHANGE occurs.
Static measurement (Non-Realtime)	NMXNOTIFY_NOTIFY	0x000	The static data inside the DLL has been updated.

There are two technical ways for receiving notifications:

- Windows Messages, see [NMX_RegisterMessage_1](#)¹⁹⁴, or
- Callback Functions, see [NMX_RegisterCallback_1](#)¹⁹⁶.

If technically possible in your application, Messages are recommended. Use Callbacks, if it is difficult or impossible to read the Windows message queue.

Inside the NMX DLL, notifications are send/called from a separate thread. An advantage of this is, that the communication thread is not interrupted, if message or callback handling takes too much time.

Nevertheless it is very important that the notification itself and reading, processing and displaying data is handled in a separate threads. The following approach is suggested:

- In case a notification occurs, set a global flag.
- Check this flag cyclically in the same thread, which handles the DLL function calls. This cyclic check can for example be implemented into a 30ms timer-event of the GUI.

Further comments:

- All notifications are automatically removed, when a connection is closed. You will have to re-register them after establishing a new connection.
- It is possible to combine multiple notifications to the same Windows Message or Callback. Register each of the Notifications, which shall be combined, using the same Windows Message or Callback Function.
- If you have multiple connections, the notifications must be registered for each connection separately.

4.3.7.1 NMX_RegisterMessage_1

This function is used to register a notification message at the DLL.

Definition

```
NMX_STATUS NMX_RegisterMessage_1(  
    NMX_PHANDLE pHandle,  
    unsigned long ulNotification,  
    HWND hWnd,  
    unsigned long ulMsgCode,
```

```
WPARAM tWParam,  
LPARAM tLParam);
```

Parameter

pHandle

[Connection Handle](#)¹⁷⁸

ulNotification

[Notification type](#)¹⁹², e.g. NMXNOTIFY_NEW_STATIC32.

hWnd

Window Handle of the Windows Window, which handles the message queue.

ulMsgCode

Message number, which must be defined by the application (see also comment below).

tWParam

The wParam of the Windows message.

tLParam

The lParam of the Windows message.

Typical function call (C example)

```
NMX_RegisterMessage_1(pHandle, NMXNOTIFY_NEW_STATIC32,  
static_cast<HWND>(Handle.ToPointer()), WM_MESSAGE_NEW_STATIC32,  
0, 0);
```

[.Net DLL](#)¹⁶⁸ specific implementation

```
NMX_MSTATUS RegisterMessage_1(  
    System::IntPtr pHandle,  
    NMX_NOTIFICATION eNotification,  
    System::IntPtr hWnd,
```

```
System::UInt32 ulMsgCode,
System::UInt32 twParam,
System::UInt32 tlParam);
```

`NMX_NOTIFICATION` is an enum type, which is defined in the .Net - DLL. The available notifications are identical to the [standard notifications](#)^[192], except that they don't have a fixed binary representation.

Comments

If `hWnd` is `NULL`, a registered message will be cleared.

Message Code (`ulMsgCode`)

The message-number is defined by your application. Using Visual C++, this can be done for example as follows:

```
#define WM_MESSAGE_DISCONNECTED (WM_USER +
NMXNOTIFY_DISCONNECTED)
#define WM_MESSAGE_FAILURE_DATA_EXCHANGE (WM_USER +
NMXNOTIFY_FAILURE_DATA_EXCHANGE)
#define WM_MESSAGE_RECONNECTED (WM_USER +
NMXNOTIFY_RECONNECTED)
#define WM_MESSAGE_NEW_STATIC32 (WM_USER +
NMXNOTIFY_NEW_STATIC32)
#define WM_MESSAGE_SAMPLING_NEWDATA (WM_USER +
NMXNOTIFY_SAMPLING_NEW_DATA)
#define WM_MESSAGE_SAMPLING_ALLRECEIVED (WM_USER +
NMXNOTIFY_SAMPLING_ALL_DATA_RECEIVED)
#define WM_MESSAGE_SAMPLING_FINISHED (WM_USER +
NMXNOTIFY_SAMPLING_FINISHED)
#define WM_MESSAGE_SAMPLING_ERROR (WM_USER +
NMXNOTIFY_SAMPLING_ERROR)
#define WM_MESSAGE_SAMPLING_BUFFER_OVERFLOW (WM_USER +
NMXNOTIFY_SAMPLING_BUFFER_OVERFLOW)
#define WM_MESSAGE_SAMPLING_TIMEOUT (WM_USER +
NMXNOTIFY_SAMPLING_TIMEOUT)
```

4.3.7.2 NMX_RegisterCallback_1

This function is used to register a notification callback function at the DLL.

Definition

```
NMX_STATUS NMX_RegisterCallback_1(
```

```
    NMX_PHANDLE pHandle,  
    unsigned long ulNotification,  
    NMX_NOTIFICATION_CALLBACK* pCbFunction,  
    void* pvContext);
```

Parameter

pHandle

[Connection Handle](#)¹⁷⁸

ulNotification

[Notification type](#)¹⁹², e.g. NMXNOTIFY_NEW_STATIC32.

pCbFunction

Pointer to callback function.

pvContext

A caller provided context pointer which is passed unchanged to the callback function.

Typical function call (C example)

```
RegisterCallback_1(pHandle, NMXNOTIFY_NEW_STATIC32,  
&Callback_NewStatic32, (void*)&ulCallbackContext);
```

Comments

If pCbFunction is NULL, a registered callback will be cleared.

Callback function

This function is the prototype for a callback notification.

Definition

```
void NMX_CALLCONV NMX_NOTIFICATION_CALLBACK(IN void*  
pvContext);
```

Parameter

pvContext

This parameter is the same which was passed to the function `NMX_RegisterCallback_1`. The application can store a context information in this pointer.

[.Net DLL](#)^[168] specific implementation

```
NMX_MSTATUS RegisterCallback_1(  
    System::IntPtr pHandle,  
    NMX_NOTIFICATION eNotification,  
    OnNotification^ NotificationDelegate);
```

`NMX_NOTIFICATION` is an enum type, which is defined in the .Net - DLL. The available notifications are identical to the [standard notifications](#)^[192], except that they don't have a fixed binary representation.

In .Net, delegates are used instead of function pointers. The .Net - DLL calls this delegates in case of a notification.

In C#, an example for a delegate function is:

```
public void OnNewStaticData(IntPtr pHandle)  
{  
}
```

The handle of the calling DLL-instance is provided as a function parameter. If you have only one instance (= 1 connection to a device), don't care about it. This is the most common case.

In case you have multiple instances, the handle can be used to distinguish between multiple connections.

An example for registering the delegate is:

```
cNmx.RegisterCallback_1(pDevice, NMX_NOTIFICATION.NEW_STATIC32,  
OnNewStaticData);
```

Note: Since the delegate is directly forwarded to the unmanaged DLL, the .Net wrapper fixes the delagte pointer in the garbage collector (gc.Alloc).

Comments

This function is called in a different thread context. The application must handle the synchronisation.

4.3.8 Get device information

Several functions are available to get information about the measurement system.

Using this information is optional.

The following is available:

- Information about [measurement boxes](#)^[200] (digital type plate).
- Information about [measurement channels](#)^[206] (type information and digital type plate of probe).
- Information about digital [inputs](#)^[211] & [outputs](#)^[213].

4.3.8.1 NMX_GetBoxCount_1

This function is used to read the number of measurement boxes, which the device has.

Definition

```
NMX_STATUS NMX_GetBoxCount_1(  
    NMX_PHANDLE pHandle,  
    unsigned long* pulBoxCount);
```

Parameter

pHandle

[Connection Handle](#)^[178]

pulBoxCount

Number of measurement boxes, which the system has.

Typical function call (C example)

```
NMX_GetBoxCount_1(pHandle, &ulBoxCount);
```

[.Net DLL](#)¹⁶⁶ specific implementation

```
NMX_MSTATUS GetBoxCount_1(System::IntPtr pHandle,  
System::UInt32 %pulBoxCount);
```

Comments

pulBoxCount only returns a value, if the function return code is NST_SUCCESS. Otherwise the value remains unchanged.

4.3.8.2 NMX_GetBoxInfo_1

This function is used to read information about a measurement box (digital type plate).

Definition

```
NMX_STATUS NMX_GetBoxInfo_1(  
    NMX_PHANDLE pHandle,  
    unsigned long ulBoxNo,  
    unsigned long* pulInfoData, unsigned long  
ulInfoDataNElements,  
    unsigned long long* pudMacAddress,  
    char* pcSerNo, unsigned long ulSizeofSerNo,  
    char* pcProdCode, unsigned long ulSizeofProdCode,  
    char* pcOrderNo, unsigned long ulSizeofOrderNo,  
    char* pcName, unsigned long ulSizeofName);
```

Parameter

pHandle

[Connection Handle](#)¹⁷⁸

ulBoxNo

Number of the measurement box, for which the information shall be provided.

The first measurement box has the number 0.

In a system with 5 measurement boxes, these are numbered 0..4.

pullInfoData

Pointer to an array of unsigned 32 Bit values.

To avoid a large amount of function parameters, several information will be stored in this array.

See below for more information about the array content.

An array size of 32 elements is recommended. Initialize these with 0.

ulInfoDataNElements

Number of elements of the array, to which pullInfoData points.

Example: The array has 32 elements, its size is 128 Bytes. Then:

```
ulInfoDataNElements = 32;
```

puDMacAddress

MAC address of the box.

pcSerNo

ASCII based string with serial number of the box. The maximum string length is 17 characters (16 + Termination).

ulSizeofSerNo

Maximum size of pcSerNo in Bytes/Characters.

pcProdCode

ASCII based string with production code of the box. The maximum string length is 17 characters (16 + Termination).

ulSizeofProdCode

Maximum size of pcProdCode in Bytes/Characters.

pcOrderNo

ASCII based string with order number of the box. The maximum string length is 33 characters (32 + Termination).

ulSizeofOrderNo

Maximum size of pcOrderNo in Bytes/Characters.

pcName

ASCII based string with name of the box. The maximum string length is 129 characters (128 + Termination).

ulSizeofName

Maximum size of pcName in Bytes/Characters.

Typical function call (C example)

```
NMX_GetBoxInfo_1(pHandle, ulBoxNo, aulInfoData,  
sizeof(aulInfoData) / 4, &udMacAddress, acSerNo,  
sizeof(acSerNo), acProdCode, sizeof(acProdCode), acOrderNo,  
sizeof(acOrderNo), acName, sizeof(acName));
```

Content of the array pullInfoData

Element	Content	Description
0	Box number	Number of the box.
1	Hardware Version Major	Version of the electronics.
2	Hardware Version Minor	
3	Hardware Revision	Compatibility code between the hardware and the firmware. It ensures that a firmware update is only allowed if the firmware version is compatible to the hardware revision.
4	Firmware Version Major	Firmware version of the box. The first part of the firmware version is incremented in case of major changes.
5	Firmware Version Minor	The second part of the firmware version is incremented in case new functionality has been implemented.
6	Firmware Version Patch	The third part of the firmware version is incremented in case one or more bugs were fixes.
7	Firmware Version Build	The fourth part of the firmware version is an internal counter.
8	Number of measurement channels	Total number of measurement channels.
9	Number of 64 Bit measurement channels	Number of 64 Bit measurement channels, which the box has.
10	Number of 32 Bit measurement channels	Number of 32 Bit measurement channels, which the box has.
11	Number of 16 Bit measurement channels	Number of 16 Bit measurement channels, which the box has.
12	Number of 8 Bit measurement channels	Number of 8 Bit measurement channels, which the box has.
13	Number of digital inputs	Total number of digital input bits. For data readout: <ul style="list-style-type: none"> • The number of digital inputs is always rounded up to a multiple of 8. If for example 2 digital inputs are available. These are rounded up to 8, whereas the inputs 3-8 are always low. • Each 8 Bits are combined in 1 Byte

[.Net DLL](#)^[166] specific implementation

```

NMX_MSTATUS GetBoxInfo_1(
    System::IntPtr pHandle,
    System::UInt32 ulBoxNo,
    array<System::UInt32>^aulInfoData,
    System::UInt64 %pudMacAddress,
    System::String ^%strSerial,
    System::String ^%strProdCode,
    System::String ^%strOrderNo,
    System::String ^%strName);

```

No Length/Sizeof-Parameters are required for the array aullInfoData and the strings, since this information is not required in a .Net environment.

For aullInfoData a size of 32 elements is recommended. The strings strSerial, strProdCode, strOrderNo and strName are directly returned as Unicode-strings. Hence no additional conversion is required.

Comments

It is good practice [reading the box count](#)^[199] first. Then read the box information for each box.

The function only returns values, if the function return code is NST_SUCCESS. Otherwise the values remain unchanged.

4.3.8.3 NMX_UpdateChannelInfo_1

This function forces the NMX DLL to re-read measurement channel information..

Definition

```

NMX_STATUS NMX_UpdateChannelInfo_1(
    NMX_PHANDLE pHandle);

```

Parameter

pHandle

[Connection Handle](#)^[178]

Typical function call (C example)

```
NMX_UpdateChannelInfo_1(pHandle);
```

[.Net DLL](#)^[168] specific implementation

```
NMX_MSTATUS UpdateChannelInfo_1(System::IntPtr pHandle);
```

Comments

The NMX DLL reads the channel information from the device once after connecting. In case the information at the device side changes afterwards, e.g. due to connecting a different probe, the data within the NMX DLL is outdated.

Call this function to update this data.

4.3.8.4 NMX_GetChannelCount_1

This function is used to read the number of measurement channels and the number of digital in-/outputs, which the device has.

Definition

```
NMX_STATUS NMX_GetChannelCount_1(  
    NMX_PHANDLE pHandle,  
    unsigned long* pulChannelCount,  
    unsigned long* pulNDigitalInputBytes,  
    unsigned long* pulNDigitalOutputBytes);
```

Parameter

pHandle

[Connection Handle](#)^[178]

pulChannelCount

Number of measurement channels, which the system has.

pulNDigitalInputBytes

Number of digital input bytes, which the system has.
(= Number of digital inputs / 8)

pulNDigitalOutputBytes

Number of digital output bytes, which the system has.
(= Number of digital outputs / 8)

Typical function call (C example)

```

NMX_GetChannelCount_1(pHandle, &ulChannelCount,
&ulNDigiInBytes, &ulNDigiOutBytes);

```

[.Net DLL](#) ¹⁶⁸ specific implementation

```

NMX_MSTATUS GetChannelCount_1(
    System::IntPtr pHandle,
    System::UInt32 %pulChannelCount,
    System::UInt32 %pulNDigitalInputBytes,
    System::UInt32 %pulNDigitalOutputBytes);

```

Comments

The function only returns values, if the function return code is NST_SUCCESS. Otherwise the values remain unchanged.

4.3.8.5 NMX_GetChannelInfo_1

This function is used to read information about a measurement channel. If supported by the probe, additional probe information is provided (digital probe type plate).

Definition

```

NMX_STATUS NMX_GetChannelInfo_1(
    NMX_PHANDLE pHandle,
    unsigned long ulChannelNo,
    unsigned long* pulChannelType,
    unsigned long* pulNDigits,
    unsigned long* pulBoxNo,

```

```
unsigned long* pulReserved,  
unsigned long* pulBoxChannelNo,  
signed long* pslRawDataType,  
signed long* pslFactNumerator,  
signed long* pslFactDenominator,  
float* pflFactDigitsToUnit,  
char* pcUnit, unsigned long ulSizeofUnit,  
char* pcOrderNo, unsigned long ulSizeofOrderNo,  
char* pcSerialNo, unsigned long ulSizeofSerialNo);
```

Parameter

pHandle

[Connection Handle](#) 

ulChannelNo

Number of the measurement channel.
The first channel has the number 0.
Having a device with 8 measurement channels, these are numbered 0..7.

pulChannelType

Measurement channel type. See below for a list of channel types.

pulNDigits

Recommended max. number of decimal places for the measurement value, converted to its unit.
E.g. `pulNDigits = 2` --> 54.**17**µm

pulBoxNo

Number of the Box, where the measurement channel is located.

pulReserved

Reserved for future use.

pulBoxChannelNo

Number of the channel within its measurement box.
Numbering starts at 1 with each Box.

Example: In a system with 2 Boxes à 8 Channels, Channel number 11 is the 4th channel of the second box. Thus the value 4 is returned.

pslRawDataType

Data type, in which the measurement data is acquired.

0 = Unknown (DTRAW_UNKNOWN)

1 = signed 8 Bit (DTRAW_SINT08)

2 = signed 16 Bit (DTRAW_SINT16)

4 = signed 32 Bit (DTRAW_SINT32)

8 = signed 64 Bit (DTRAW_SINT64)

See the [data types](#)^[164] chapter for more information.

pslFactNumerator

Numerator of the factor, which is used for converting the digit value into its physical unit.

For some channel types, the physical unit is unknown by the device.

Then this value is 1.

For some channel types, e.g. inductive probes, this is also the maximum stroke, which the probe has.

pslFactDenominator

Denominator of the factor, which is used for converting the digit value into its physical unit.

For some channel types, the physical unit is unknown by the device.

Then this value is 1.

pflFactDigitsToUnit

$$= \text{pslFactNumerator} / \text{pslFactDenominator}.$$

Floating point representation of the factor, which is used for converting the digit value into its physical unit.

(ANSI/IEEE Std 754-1985; IEC-60559:1989: single precision 32 Bit)

pcUnit

ASCII based string with the channels physical unit (or "Digits" if the unit is unknown). The maximum string length is 9 characters (8 + Termination).

ulSizeofUnit

Maximum size of pcUnit in Bytes/Characters.

pcOrderNo

ASCII based string with the probe/sensors order number, if available.
The maximum string length is 17 characters (16 + Termination).

ulSizeofOrderNo

Maximum size of pcOrderNo in Bytes/Characters.

pcSerialNo

ASCII based string with the probe/sensors serial number, if available.
The maximum string length is 17 characters (16 + Termination).

ulSizeofSerialNo

Maximum size of pcSerialNo in Bytes/Characters.

Typical function call (C example)

```
NMX_GetChannelInfo_1(pHandle, ulChannel, &ulChannelType,  
&ulNDigits, &ulBoxNo, NULL, &ulBoxChannelNo, &slRawDataType,  
&slFactNumerator, &slFactDenominator, &flFactDigitsToUnit,  
acUnit, sizeof(acUnit), acOrderNo, sizeof(acOrderNo),  
acSerialNo, sizeof(acSerialNo));
```

Channel types

The channel type allows identifying the channel and/or probe type. The list is constantly expanded. Currently the following types are supported:

Type	Hex value	Description
CHANNEL_TYPE_U	0x0000	No channel type available
CHANNEL_TYPE_T	0x0100	Tesa HalfBridge or compatible, 13 kHz, 3Veff, Default Stroke: ±2mm at 73.75mV/V/mm
CHANNEL_TYPE_T	0x0101	Tesa HalfBridge or compatible, 13 kHz, 3Veff, Linearized, Max Stroke ±2mm
CHANNEL_TYPE_T	0x0103	Tesa HalfBridge or compatible, 13 kHz, 3Veff, Linearized, Max. Stroke ±3mm
CHANNEL_TYPE_T	0x0105	Tesa HalfBridge or compatible, 13 kHz, 3Veff, Linearized, Max. Stroke ±5mm
CHANNEL_TYPE_IE	0x0110	Knäbel IET, 50 kHz, 1.5Veff, Default Stroke: ±200µm
CHANNEL_TYPE_S	0x0120	Solartron LVDT or compatible, 5 kHz, 3Veff, Default Stroke: ±1mm at 200mV/V/mm
CHANNEL_TYPE_S	0x0121	Solartron customized type.
CHANNEL_TYPE_F	0x0130	Feinprüf / Mahr, 20 kHz, 3Veff, Default Stroke: ±1mm
CHANNEL_TYPE_M	0x0140	Marposs LVDT, Default Stroke: ±1mm
CHANNEL_TYPE_IN	0x0200	Incremental Encoder TTL/RS422
CHANNEL_TYPE_IN	0x0210	Incremental Encoder 1Vpp
CHANNEL_TYPE_A	0x0300	Analogue Input ±10V Differential
CHANNEL_TYPE_T	0x0400	Thermocouple, K-Type
CHANNEL_TYPE_L	0x0500	Laser-Sensor (Sub-Types available).

[Net DLL](#) ¹⁶⁸ specific implementation

`NMX_MSTATUS GetChannelInfo_1(`

```

System::IntPtr pHandle,
System::UInt32 ulChannelNo,
System::UInt32 %pulChannelType,
System::UInt32 %pulNDigits,
System::UInt32 %pulBoxNo,
System::UInt32 %pulReserved,
System::UInt32 %pulBoxChannelNo,
System::Int32 %pslRawDataType,
System::Int32 %pslFactNumerator,
System::Int32 %pslFactDenominator,
System::Single %pflFactDigitsToUnit,
System::String ^%strUnit,
System::String ^%strOrderNo,
System::String ^%strSerNo);

```

No Length/Sizeof-Parameters are required for the strings, since this information is not required in a .Net environment. They are directly returned as Unicode-strings. Hence no additional conversion is required.

Comments

It is good practice [reading the channel count](#)^[205] first. Then read the channel information for each channel.

The function only returns values, if the function return code is NST_SUCCESS. Otherwise the values remain unchanged.

4.3.8.6 NMX_GetDigitalInputInfo_1

This function is used to read information about a digital input byte. Each input byte represents 8 digital input bits.

Definition

```

NMX_STATUS NMX_GetDigitalInputInfo_1(
    NMX_PHANDLE pHandle,
    unsigned long ulInputByteNo,
    unsigned long* pulBoxNo,
    unsigned long* pulBoxByteNo);

```

Parameter

pHandle

[Connection Handle](#)^[178]

ulInputByteNo

Number of the digital input byte.

The first byte has the number 0.

Example: Having a device with 4 input bytes, these are numbered 0..3.

pulBoxNo

Number of the Box, where the digital inputs are located.

pulBoxByteNo

Number of the digital input byte within its measurement box.

Numbering starts at 1 with each Box.

Example: In a system with 2 Boxes à 16 digital inputs, input byte 3 is the 1st input byte of the second box. Thus the value 1 is returned.

Typical function call (C example)

```
NMX_GetDigitalInputInfo_1(pHandle, ulInByte, &ulBoxNo,  
&ulBoxByteNo);
```

[.Net DLL](#)^[168] specific implementation

```
NMX_MSTATUS GetDigitalInputInfo_1(  
    System::IntPtr pHandle,  
    System::UInt32 ulInputByteNo,  
    System::UInt32 %pulBoxNo,  
    System::UInt32 %pulBoxByteNo);
```

Comments

It is good practice [reading the digital input count](#)^[205] first. Then read the byte information for each digital input byte.

The function only returns values, if the function return code is NST_SUCCESS. Otherwise the values remain unchanged.

4.3.8.7 NMX_GetDigitalOutputInfo_1

This function is used to read information about a digital output byte. Each output byte represents 8 digital output bits.

Definition

```
NMX_STATUS NMX_CALLCONV NMX_GetDigitalOutputInfo_1(  
    NMX_PHANDLE pHandle,  
    unsigned long ulOutputByteNo,  
    unsigned long* pulBoxNo,  
    unsigned long* pulBoxByteNo);
```

Parameter

pHandle

[Connection Handle](#) ₁₇₈

ulOutputByteNo

Number of the digital output byte.
The first byte has the number 0.
Example: Having a device with 4 output bytes, these are numbered 0..3.

pulBoxNo

Number of the Box, where the digital outputs are located.

pulBoxByteNo

Number of the digital output byte within its measurement box.
Numbering starts at 1 with each Box.

Example: In a system with 2 Boxes à 16 digital outputs, output byte 3 is the 1st output byte of the second box. Thus the value 1 is returned.

Typical function call (C example)

```
NMX_GetDigitalOutputInfo_1(pHandle, ulOutByte, &ulBoxNo,  
&ulBoxByteNo);
```

[.Net DLL](#)^[166] specific implementation

```

NMX_MSTATUS GetDigitalOutputInfo_1(
    System::IntPtr pHandle,
    System::UInt32 ulOutputByteNo,
    System::UInt32 %pulBoxNo,
    System::UInt32 %pulBoxByteNo);

```

Comments

It is good practice [reading the digital output count](#)^[205] first. Then read the byte information for each digital output byte.

The function only returns values, if the function return code is NST_SUCCESS. Otherwise the values remain unchanged.

4.3.9 Static Measurement (Non-Realtime)

Static measurement is used to get a snapshot of the current measurement values and digital input states. It is very easy to implement

For more information see the chapter "[Static vs. Sampling](#)^[156]".

4.3.9.1 NMX_StaticGet32_1

This function is used to get a snapshot of:

- Current measurement values
- Current hardware status of the measurement channels
- Current digital input status
- Current status of the measurement boxes ("Box event")

See also the [HowTo](#)^[258]-Guide.

Definition

```

NMX_STATUS NMX_StaticGet32_1(
    NMX_PHANDLE pHandle,
    signed long aslValues[], unsigned long ulValuesNElements,
    unsigned char aucHardwareStatus[], unsigned long
ulSizeofHardwareStatus,
    unsigned char aucDigiIn[], unsigned long ulSizeofDigiIn,
    unsigned char aucBoxStatus[], unsigned long
ulSizeofBoxStatus,

```

```
unsigned long* pulUpdateCtr);
```

Parameter

pHandle

[Connection Handle](#)¹⁷⁸

aslValues

Array, in which the measurement values shall be stored.

The data type of the array is "signed 32 Bit". See chapter "[Data Types](#)¹⁶⁴" for more information.

The array must be provided by your application.

Technically, this parameter is the same as a pointer to the array. Its type is: `signed long*`

ulValuesNElements

Number of elements of the array `aslValues`.

Example: `aslValues` points to an array with 64 elements, each with 32 Bit (Total size = 256 Bytes). Then: `ulValuesNElements = 64;`

aucHardwareStatus

Array, in which the hardware status for the measurement channels shall be stored.

The data type of the array is "unsigned 8 Bit".

The array must be provided by your application.

Technically, this parameter is the same as a pointer to the array. Its type is: `unsigned char*`

ulSizeofHardwareStatus

Size of the array `aucHardwareStatus` in Bytes.

aucDigiln

Array, in which the digital input bytes shall be stored.

The data type of the array is "unsigned 8 Bit".

The array must be provided by your application.

Technically, this parameter is the same as a pointer to the array. Its type is: `unsigned char*`

ulSizeofDigiIn

Size of the array aucDigiIn in Bytes.

aucBoxStatus

Array, in which the status information for each box shall be stored.

The data type of the array is "unsigned 8 Bit".

The array must be provided by your application.

Technically, this parameter is the same as a pointer to the array. Its type

is: `unsigned char*`

ulSizeofBoxStatus

Size of the array aucBoxStatus in Bytes.

pulUpdateCtr

A DLL-internal counter is updated each time new static data has been received from the device. This counter can be read-out here.

Typical function call (C example)

```

NMX_StaticGet32_1(pHandle, aslMeasVal, sizeof(aslMeasVal)/4,
aucHardStat, sizeof(aucHardStat), aucDigiIn, sizeof(aucDigiIn),
aucBoxStatus, sizeof(aucBoxStatus), &ulNUpdates);

```

[.Net DLL](#) ¹⁶⁸ specific implementation

```

NMX_MSTATUS StaticGet32_1(
    System::IntPtr pHandle,
    array<System::Int32>^aslValues,
    array<System::Byte>^aucHardwareStatus,
    array<System::Byte>^aucDigiIn,
    array<System::Byte>^aucBoxStatus,
    System::UInt32 %pulUpdateCtr);

```

No Length/Sizeof-Parameters are required for the arrays, since this information automatically available in a .Net environment.

The arrays will not be resized within the function call for performance reasons. This means they should be large enough to store all the data.

Example: in a system with 24 measurement channels, the arrays aslValues and aucHardwareStatus should have a minimum size of 24 array elements.

If the array is shorter, not all data will be available for your application. (The size is checked -> no risk for crash.)

Comments

The [notification](#)^[192] NMXNOTIFY_NEW_STATIC32 can be used to get informed about new data.

Never use digital in-/outputs for emergency critical applications. If required, use an external emergency circuit. See the users manual of your measurement system for more information.

Status-Byte for measurement channels for 1Vpp incremental encoders (INC)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
PwrOvl d		Refmar k	Vector	GCom p	OCom p	AmpErr	Fast

PwrOvl

Error: A power supply overload of the incremental encoder(s) has been detected.

Refmark

The reference index has been crossed.

Vector

Error: The signal vector, which has been calculated from the cosine- and sine-signal, is too small. (Can only occur with 1Vpp incremental encoders.)

GComp

Error: The gain-control has reached its limit. (Can only occur with 1Vpp incremental encoders.)

OComp

Error: The offset-control has reached its limit. (Can only occur with 1Vpp incremental encoders.)

AmpErr

Error: One or both AD-converters for the measurement of the sine-/cosine-signal is/are overdriven. (Can only occur with 1Vpp incremental encoders.)

Fast

Error: The input frequency is too high.

Status-Byte for inductive probes (TFV)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
LinActive	IdentActive		LinFailure				ShortCirc

ShortCirc

Error: Short circuit of the sine-oscillator

LinFailure

Failed reading / applying the linearisation data.
This bit is only available, if the measurement hardware and the probe support linearisation.

IdentActive

Reading the probe identification and linearisation data is active.
This bit is only available, if the measurement hardware and the probe support linearisation.

LinActive

Probe linearisation is active.
This bit is only available, if the measurement hardware and the probe support linearisation.

Status-Byte for analogue inputs (AIN)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
24VOvl d	VRefOv ld						

24VOvld

Error: Overload of the 24V output supply.

VRefOvld

Error: Overload of the reference voltage output.

Status-Byte for temperature measurement (TEMP)

Bit 7	Bit 6	Bit 5	Bit 4	Bit 3	Bit 2	Bit 1	Bit 0
TempI nvalid	SenseF ailure	ADErro r	ColdJu nctionE rrorHw	ColdJu nctionE rrorSw	SenseV oltHigh	SenseV oltLow	OutOfR ange

OutOfRange

Sensor input voltage is out of range

SenseVoltLow

Sensor reading is below normal range.

SenseVoltHigh

Sensor reading is above normal range.

ColdJunctionErrorSw

Cold junction sensor result is beyond normal range.

ColdJunctionErrorHw

Cold junction sensor has a hardware fault error.

ADError

Bad ADC reading. Could be large external noise event.

SenseFailure

Bad sensor reading.

TempInvalid

Temperature value may be invalid.

4.3.9.2 NMX_StaticSetMedianDepth_1

This function is used to modify, enable or disable the median filter, which is applied on static measurement values.

Definition

```
NMX_STATUS NMX_StaticSetMedianDepth_1(  
    NMX_PHANDLE pHandle,  
    unsigned long ulMedianDepth);
```

Parameter

pHandle

[Connection Handle](#)¹⁷⁸

ulMedianDepth

Number of samples, which are used to calculate their median value.
Using an odd value (3, 5, 7, ...) is recommended.
A value of 1 disables the median calculation.

Typical function call (C example)

```
NMX_StaticSetMedianDepth_1(pHandle, 3);
```

[.Net DLL](#)¹⁶⁸ specific implementation

```
NMX_MSTATUS StaticSetMedianDepth_1(  
    System::IntPtr pHandle,  
    System::UInt32 ulMedianDepth);
```

Comments

The default depth is 5.

The value should not be too high. Otherwise you get a long delay.

4.3.9.3 NMX_SetOutputs_1

This function is used to change the state of the digital outputs.

Definition

```
NMX_STATUS NMX_SetOutputs_1(  
    NMX_PHANDLE pHandle,  
    unsigned char* pucDigiOut, unsigned long ulSizeofDigiOut,  
    unsigned char ucForceSendImmediately);
```

Parameter

pHandle

[Connection Handle](#) 

pucDigiOut

Pointer to the array, in which the digital output bytes are stored.

ulSizeofDigiOut

Size of the array "behind" pucDigiOut in Bytes.

ucForceSendImmediately

0 -> Default: The output data will be send with the next communication cycle. The default cycle time is approximately 30ms.

1 -> Send the output data immediately. Use this only, if an urgent update is required. Sending urgent updates repeatedly at a high frequency could slow down transfer rate for sampled data.

Typical function call (C example)

```
NMX_SetOutputs_1(pHandle, aucDigiOut, sizeof(aucDigiOut), 0);
```

[.Net DLL](#)^[168] specific implementation

```
NMX_MSTATUS SetOutputs_1(  
    System::IntPtr pHandle,  
    array<System::Byte>^aucDigiOut,  
    System::Byte ucForceSendImmediately);
```

No Length/Sizeof-Parameter is required for the array aucDigiOut, since this information automatically available in a .Net environment.

Comments

With ucForceSendImmediately = 0, the output data is only updated inside the NMX DLL with this function call. Each time cyclic data is exchanged between the DLL and the device, the output data is send to the device. No matter if it has been changed or not.

Never use digital in-/outputs for emergency critical applications. If required, use an external emergency circuit. See the users manual of your measurement system for more information.

4.3.9.4 NMX_DisableOutputUpdate_1

Digital output data is normally transferred cyclically from the NMX DLL to the device. This function is used to stop this cyclic update.

It is typically not used within standard measurement applications. It could be helpful, if a manual manipulation of the outputs shall be done, e.g. using the configuration tool of the measurement system.

Definition

```
NMX_STATUS NMX_DisableOutputUpdate_1(  
    NMX_PHANDLE pHandle);
```

Parameter

pHandle

[Connection Handle](#)¹⁷⁸

Typical function call (C example)

```
NMX_DisableOutputUpdate_1(pHandle);
```

[.Net DLL](#)¹⁶⁸ specific implementation

```
NMX_MSTATUS DisableOutputUpdate_1(System::IntPtr pHandle);
```

4.3.9.5 NMX_DigitalIoConfig_1

This function is used to disable or enable the automatic reset of digital outputs after a communication timeout.

By default this is enabled, which means that all outputs are set to low, if there is a breakdown of the communication between the NMX DLL and the digital output.

Definition

```
NMX_STATUS NMX_DigitalIoConfig_1(  
    NMX_PHANDLE pHandle,  
    unsigned char ucOutputResetEnabled);
```

Parameter

pHandle

[Connection Handle](#)¹⁷⁸

ucOutputResetEnabled

0 -> Resetting the digital outputs to low state after communication breakdown is disabled.

1 -> Default: Resetting the digital outputs to low state after communication breakdown is enabled.

Typical function call (C example)

```
NMX_DigitalIoConfig_1(pHandle, 0);
```

[.Net DLL](#)^[168] specific implementation

```
NMX_MSTATUS DigitalIoConfig_1(  
    System::IntPtr pHandle,  
    System::Byte ucOutputResetEnabled);
```

Comments

By default, the state of all digital outputs is automatically set to low after communication breakdown is enabled. This means: if the communication breaks down: all outputs are set to low.

4.3.9.6 NMX_DigitalOutputsGetState_1

This function allows reading the current state of all digital outputs from the device.

Definition

```
NMX_STATUS NMX_DigitalOutputsGetState_1(  
    NMX_PHANDLE pHandle,  
    unsigned char* pucOutputState,  
    unsigned long ulSizeofOutputState);
```

Parameter

pHandle

[Connection Handle](#)^[178]

pucOutputState

Pointer to the array, in which the digital output bytes shall be stored.
The data type of the array is "unsigned 8 Bit".
The array must be provided by your application.

ulSizeofOutputState

Size of pucOutputState in Bytes.

Typical function call (C example)

```
NMX_DigitalOutputsGetState_1(pHandle, aucOutputs,  
sizeof(aucOutputs));
```

[.Net DLL](#)^[168] specific implementation

```
NMX_MSTATUS DigitalOutputsGetState_1(  
    System::IntPtr pHandle,  
    array<System::Byte>^aucOutputState);
```

No Length/Sizeof-Parameter is required for the array aucOutputState, since this information automatically available in a .Net environment.

The array will not be resized within the function call for performance reasons. This means it should be large enough to store all the data.

Example: in a system with 32 digital outputs (= 4 output bytes), the array aucOutputState should have a minimum size of 4 array elements.

If the array is shorter, not all data will be available for your application. (The size is checked -> no risk for crash.)

Comments

The purpose of this function is getting the state of all digital outputs once after establishing a connection. It should not be called cyclically.

The state reflects the internal data inside the measurement system. It does not reflect the physical state of an output. Usually both are the same, but under fault conditions they may differ.

4.3.10 Sampling LowLevel (Time-Triggered Realtime Measurement)

Sampling is used to get real-time data from the measurement system. This covers measurement values as well as digital in-/output states.

For more information see the chapter "[Static vs. Sampling](#)"^[156].

4.3.10.1 NMX_Sampling_GetMaxSpeed_1

This function is used to get the maximum possible sampling speed.

Definition

```
NMX_STATUS NMX_Sampling_GetMaxSpeed_1(  

```

```
    NMX_PHANDLE pHandle,  
    unsigned long *pulInSamplePeriodUs);
```

Parameter

pHandle

[Connection Handle](#)^[178]

pullnSamplePeriodUs

Minimum sample period in μ s. The maximum sampling speed is the reciprocal value.

Typical function call (C example)

```
NMX_Sampling_GetMaxSpeed_1(pHandle, &ulMinSamplePeriod);
```

[.Net DLL](#)^[168] specific implementation

```
NMX_MSTATUS Sampling_GetMaxSpeed_1(  
    System::IntPtr pHandle,  
    System::UInt32 %pulInSamplePeriodUs);
```

Comments

The maximum possible speed for endless sampling may be slower. Consult the users manual of the measurement system for more information about sampling speed.

4.3.10.2 NMX_Sampling_Reset_1

Reset the whole sampling configuration. If sampling is active, stop it.

This function must be called before a new list of sampling elements is created via NMX_Sampling_Add...

See also the [HowTo](#)^[266]-Guide.

Definition

```
NMX_STATUS NMX_Sampling_Reset_1(  
    NMX_PHANDLE pHandle,  
    unsigned long *pulInSamplePeriodUs);
```

```
    NMX_PHANDLE pHandle);
```

Parameter

pHandle

[Connection Handle](#)^[178]

Typical function call (C example)

```
NMX_Sampling_Reset_1(pHandle, &ulMinSamplePeriod);
```

.Net DLL^[168] specific implementation

```
NMX_MSTATUS Sampling_Reset_1(System::IntPtr pHandle);
```

Comments

If an active sampling is stopped via NMX_Sampling_Stop_1 and then shall be restarted with the same sampling elements, there is no need for calling NMX_Sampling_Reset_1.

However, if a new list of sampling elements shall be created, an existing list must be cleared by calling NMX_Sampling_Reset_1.

4.3.10.3 NMX_Sampling_AddChannelsAll_1

Use this function to add all measurement channels to the list of sampling elements.

See also the [HowTo](#)^[266]-Guide.

Definition

```
NMX_STATUS NMX_Sampling_AddChannelsAll_1(  
    NMX_PHANDLE pHandle,  
    unsigned long* pulNElementsTotal);
```

Parameter

pHandle

[Connection Handle](#)^[178]

pulNElementsTotal

Total number of sampling elements, which have already been added since the last call of [NMX_Sampling_Reset_1](#)^[226].

Typical function call (C example)

```
NMX_Sampling_AddChannelsAll_1(pHandle, &ulNElements);
```

[.Net DLL](#)^[168] specific implementation

```
NMX_MSTATUS Sampling_AddChannelsAll_1(  
    System::IntPtr pHandle,  
    System::UInt32 %pulNElementsTotal);
```

Comments

It is also possible to add only a selection of measurement channels to the list of sampling elements (see [NMX_Sampling_AddChannel_1](#)^[228]).

This may

- allow for higher sampling speed with endless sampling or
- shorten the transfer time of the samples with time-limited sampling.

Please note: the same measurement channel cannot be added twice. Thus you can either use `NMX_Sampling_AddChannelsAll_1` or `NMX_Sampling_AddChannel_1`, but not both of them.

4.3.10.4 NMX_Sampling_AddChannel_1

Use this function to add a single measurement channel to the list of sampling elements.

See also the [HowTo](#)^[266]-Guide.

Definition

```
NMX_STATUS NMX_Sampling_AddChannel_1(  
    NMX_PHANDLE pHandle,  
    unsigned long ulChannelNumber,  
    unsigned long* pulNElementsTotal);
```

Parameter

pHandle

[Connection Handle](#)¹⁷⁸

ulChannelNumber

Number of the measurement channel, which shall be added.
The first channel has the number 0.
Having a device with 24 measurement channels, these are numbered 0..23.

pulNElementsTotal

Total number of sampling elements, which have already been added since the last call of [NMX_Sampling_Reset_1](#)²²⁶.

Typical function call (C example)

```
NMX_Sampling_AddChannel_1(pHandle, 5, &ulNElements);
```

[.Net DLL](#)¹⁶⁸ specific implementation

```
NMX_MSTATUS Sampling_AddChannel_1(  
    System::IntPtr pHandle,  
    System::UInt32 ulChannelNumber,  
    System::UInt32 %pulNElementsTotal);
```

Comments

It is also possible to add all measurement channels to the list of sampling elements (see [NMX_Sampling_AddChannelsAll_1](#)²²⁷).

Please note: the same measurement channel cannot be added twice. Thus you can either use `NMX_Sampling_AddChannelsAll_1` or `NMX_Sampling_AddChannel_1`, but not both of them.

4.3.10.5 NMX_Sampling_AddDigiInAll_1

Use this function to add all digital input bytes to the list of sampling elements.

See also the [HowTo^{\[266\]}](#)-Guide.

Definition

```
NMX_STATUS NMX_Sampling_AddDigiInAll_1(
    NMX_PHANDLE pHandle,
    unsigned long* pulNElementsTotal);
```

Parameter

pHandle

[Connection Handle^{\[178\]}](#)

pulNElementsTotal

Total number of sampling elements, which have already been added since the last call of [NMX_Sampling_Reset_1^{\[226\]}](#).

Typical function call (C example)

```
NMX_Sampling_AddDigiInAll_1(pHandle, &ulNElements);
```

[.Net DLL^{\[168\]}](#) specific implementation

```
NMX_MSTATUS Sampling_AddDigiInAll_1(
    System::IntPtr pHandle,
    System::UInt32 %pulNElementsTotal);
```

Comments

It is also possible to add only a selection of digital input bytes to the list of sampling elements (see [NMX_Sampling_AddDigiInByte_1^{\[231\]}](#)).

This may

- allow for higher sampling speed with endless sampling or

- shorten the transfer time of the samples with time-limited sampling.

Please note: the same digital input byte cannot be added twice. Thus you can either use `NMX_Sampling_AddDigiInAll_1` or `NMX_Sampling_AddDigiInByte_1`, but not both of them.

4.3.10.6 NMX_Sampling_AddDigiInByte_1

Use this function to add a single digital input byte to the list of sampling elements.

See also the [HowTo²⁶⁶](#)-Guide.

Definition

```
NMX_STATUS NMX_Sampling_AddDigiInByte_1(  
    NMX_PHANDLE pHandle,  
    unsigned long ulDigiInByteNo,  
    unsigned long* pulNElementsTotal);
```

Parameter

pHandle

[Connection Handle¹⁷⁸](#)

ulDigiInByteNo

Number of the digital input byte.

The first byte has the number 0.

Example: Having a device with 6 input bytes, these are numbered 0..5.

pulNElementsTotal

Total number of sampling elements, which have already been added since the last call of [NMX_Sampling_Reset_1²²⁶](#).

Typical function call (C example)

```
NMX_Sampling_AddDigiInByte_1(pHandle, 0, &ulNElements);
```

[.Net DLL¹⁶⁸](#) specific implementation

```
NMX_MSTATUS Sampling_AddDigiInByte_1(  
    System::IntPtr pHandle,  
    System::UInt32 ulDigiInByteNo,  
    System::UInt32 %pulNElementsTotal);
```

Comments

It is also possible to add all digital input bytes to the list of sampling elements (see [NMX_Sampling_AddDigiInAll_1](#)^[230]).

Please note: the same digital input byte cannot be added twice. Thus you can either use `NMX_Sampling_AddDigiInAll_1` or `NMX_Sampling_AddDigiInByte_1`, but not both of them.

4.3.10.7 NMX_Sampling_AddDigiOutAll_1

Use this function to add all digital output bytes to the list of sampling elements.

See also the [HowTo](#)^[266]-Guide.

Definition

```
NMX_STATUS NMX_Sampling_AddDigiOutAll_1(  
    NMX_PHANDLE pHandle,  
    unsigned long* pulNElementsTotal);
```

Parameter

pHandle

[Connection Handle](#)^[178]

pulNElementsTotal

Total number of sampling elements, which have already been added since the last call of [NMX_Sampling_Reset_1](#)^[226].

Typical function call (C example)

```
NMX_Sampling_AddDigiOutAll_1(pHandle, &ulNElements);
```

[.Net DLL](#)^[166] specific implementation

```

NMX_MSTATUS Sampling_AddDigiOutAll_1(
    System::IntPtr pHandle,
    System::UInt32 %pulNElementsTotal);

```

Comments

It is also possible to add only a selection of digital output bytes to the list of sampling elements (see [NMX_Sampling_AddDigiOutByte_1](#)^[233]).

This may

- allow for higher sampling speed with endless sampling or
- shorten the transfer time of the samples with time-limited sampling.

Please note: the same digital output byte cannot be added twice. Thus you can either use `NMX_Sampling_AddDigiOutAll_1` or `NMX_Sampling_AddDigiOutByte_1`, but not both of them.

4.3.10.8 NMX_Sampling_AddDigiOutByte_1

Use this function to add a single digital output byte to the list of sampling elements.

See also the [HowTo](#)^[266]-Guide.

Definition

```

NMX_STATUS NMX_Sampling_AddDigiOutByte_1(
    NMX_PHANDLE pHandle,
    unsigned long ulDigiOutByteNo,
    unsigned long* pulNElementsTotal);

```

Parameter

pHandle

[Connection Handle](#)^[178]

ulDigiOutByteNo

Number of the digital output byte.

The first byte has the number 0.

Example: Having a device with 6 output bytes, these are numbered 0..5.

pulNElementsTotal

Total number of sampling elements, which have already been added since the last call of [NMX_Sampling_Reset_1](#)^[226].

Typical function call (C example)

```
NMX_Sampling_AddDigiOutByte_1(pHandle, 1, &ulNElements);
```

[.Net DLL](#)^[188] specific implementation

```
NMX_MSTATUS Sampling_AddDigiOutByte_1(
    System::IntPtr pHandle,
    System::UInt32 ulDigiOutByteNo,
    System::UInt32 %pulNElementsTotal);
```

Comments

It is also possible to add all digital output bytes to the list of sampling elements (see [NMX_Sampling_AddDigiOutAll_1](#)^[232]).

Please note: the same digital output byte cannot be added twice. Thus you can either use `NMX_Sampling_AddDigiOutAll_1` or `NMX_Sampling_AddDigiOutByte_1`, but not both of them.

4.3.10.9 NMX_Sampling_PrepareTime_1

This function is used to prepare a time-based sampling. Before starting the sampling, it must be prepared.

See also the [HowTo](#)^[266]-Guide.

Definition

```
NMX_STATUS NMX_Sampling_PrepareTime_1(
    NMX_PHANDLE pHandle,
    unsigned long ulSamplePeriod,
    unsigned long ulDLLArrayLength,
    unsigned long long udMaxSamples);
```

Parameter

pHandle

[Connection Handle](#)^[178]

ulSamplePeriod

Time period between two samples in μs , e.g. 1000 for 1ms.

[See the chapter "Sampling Speed with Irinos" for more information.](#)^[157]

ulDLLArrayLength

Data received from the device is internally buffered in the DLL. This parameter specifies the internal buffer length in samples.

Example: The sampling speed is 1ms \rightarrow 1000Samples/s and the buffer shall contain a maximum of 10 seconds. Then `ulDLLArrayLength = 10000;`

udMaxSamples

Maximum number of samples, which shall be recorded.

Use 0 for endless sampling.

Example: The sampling speed is 1ms \rightarrow 1000Samples/s. Data shall be recorded for a maximum of 15 seconds. Then `udMaxSamples = 15000;`

Typical function call (C example)

Time-limited: `NMX_Sampling_PrepareTime_1(pHandle, 1000, 10000, 10000);`

Endless: `NMX_Sampling_PrepareTime_1(pHandle, 1000, 10000, 0);`

[.Net DLL](#)^[168] specific implementation

```

NMX_MSTATUS Sampling_PrepareTime_1(
    System::IntPtr pHandle,
    System::UInt32 ulSamplePeriod,
    System::UInt32 ulDLLArrayLength,
    System::UInt64 udMaxSamples);
  
```

Comments

-
- For short measurements of a few seconds, it is common practice using the same value for `ulDLLArrayLength` and for `udMaxSamples`.
 - The internal buffer is allocated in the heap memory of your application.
 - The larger the DLL-internal buffer is, the larger the memory consumption. For typical time-limited sampling, this is not a problem and the memory size will be between a few kBytes up to a few MBytes. This applies even to larger systems.
However, if you would use for example 256 measurement channels of 32 Bit size, and `ulDLLArrayLength` would be 100000, then 100 MBytes would be required. This may be too large for your heap memory.

4.3.10.10 NMX_Sampling_Start_1

This function is used to start sampling. Before starting the sampling, it must be [prepared](#)^[234].

See also the [HowTo](#)^[266]-Guide.

Definition

```
NMX_STATUS NMX_Sampling_Start_1(
    NMX_PHANDLE pHandle);
```

*Parameter***pHandle**

[Connection Handle](#)^[178]

Typical function call (C example)

```
NMX_Sampling_Start_1(pHandle);
```

.Net DLL^[168] *specific implementation*

```
NMX_MSTATUS Sampling_Start_1(System::IntPtr pHandle);
```

Comments

4.3.10.11 NMX_Sampling_Stop_1

This function is used to stop an active sampling.

Definition

```
NMX_STATUS NMX_Sampling_Stop_1(  
    NMX_PHANDLE pHandle);
```

Parameter

pHandle

[Connection Handle](#)¹⁷⁸

Typical function call (C example)

```
NMX_Sampling_Stop_1(pHandle);
```

[.Net DLL](#)¹⁶⁸ specific implementation

```
NMX_MSTATUS Sampling_Stop_1(System::IntPtr pHandle);
```

Comments

If sampling is inactive, nothing happens.

4.3.10.12 NMX_Sampling_ReadColumn32_1

Read sampled values column-wise. Column-wise means, that multiple samples of the same sampling element are read-out. This could for example be samples 0..999 from measurement channel 1.

See also the [HowTo](#)²⁸⁴-Guide.

Definition

```
NMX_STATUS NMX_Sampling_ReadColumn32_1(  
    NMX_PHANDLE pHandle,  
    signed long aslSamples[],
```

```
unsigned long ulMaxSamples,  
unsigned long ulElementNo,  
unsigned long ulDoNotDelete,  
unsigned long* pulSamplesCopied,  
unsigned long long* pudNoFirstSample);
```

Parameter

pHandle

[Connection Handle](#)¹⁷⁸

aslSamples

The measurement samples will be written into this array. For simplicity, all sampled values are provided with data type signed long. See chapter "[data types](#)¹⁶⁴" for more information.

ulMaxSamples

Maximum number of samples to read-out. This value must be \leq the total number of elements of aslSamples.

Example: aslSamples is defined as "signed long aslSamples[1000]".

Then ulMaxSamples must be \leq 1000.

ulElementNo

Number of the sampling element, starting with 0 for the first sampling element.

ulDoNotDelete

0: samples will be deleted from the DLL-internal buffer after readout.

1: samples will NOT be deleted from the DLL-internal buffer after readout.

pulSamplesCopied

Number of samples copied into aslSamples. This value will be \leq ulMaxSamples.

pudNoFirstSample

Number of the first sample in `aslSamples`, counted from the beginning of sampling, starting at 0.

Example:

The 1. time after the start of sampling, this function is called and 162 samples are read-out. `pudNoFirstSample = 0` will be returned.

The 2. time this function is called, 97 samples are read-out. `pudNoFirstSample = 162` will be returned.

The 3. time this function is called, 212 samples are read-out. `pudNoFirstSample = 259` will be returned ($162 + 97 = 259$).

Typical function call (C example)

Assuming `aslSamples` is defined as: `signed long aslSamples[10000];`

Example 1: `NMX_Sampling_ReadColumn32_1(pHandle, aslSamples, 10000, 4, 0, &ulSamplesCopied, &udNoFirstSample);`

Example 2: `NMX_Sampling_ReadColumn32_1(pHandle, &aslSamples[162], 10000-162, 4, 0, &ulSamplesCopied, &udNoFirstSample);`

[.Net DLL](#) ¹⁶⁸ specific implementation

```
NMX_MSTATUS Sampling_ReadColumn32_1(
    System::IntPtr pHandle,
    array<System::Int32>^aslSamples,
    System::UInt32 ulArrayIndex,
    System::UInt32 ulMaxSamples,
    System::UInt32 ulElementNo,
    System::UInt32 ulDoNotDelete,
    System::UInt32 %pulSamplesCopied,
    System::UInt64 %pudNoFirstSample);
```

The array `aslSamples` will not be resized within the function call for performance reasons. This means it should be large enough to store all the data.

Via the parameters `ulArrayIndex` and `ulMaxSamples`, it can be defined in which area of `aslSamples` the sampled data can be written. This is especially helpful, if the sampled data is read in multiple blocks of data (-> no additional copying required).

If the sampled data is read at once, typically `ulArrayIndex = 0` and `ulMaxSamples = aslSamples.Length`.

Comments

-
- The standard way is deleting sampled data inside the NMX DLL, after it has been read-out. This is done by setting: `ulDoNotDelete = 0;` If you want to be able to read it out more than once, it is possible to set `ulDoNotDelete = 1;`. Naturally this makes no sense for endless sampling!
 - It is possible to get informed about new data by the [notification](#)^[192] `NMXNOTIFY_SAMPLING_NEW_DATA`.
 - It is good practice reading sampled data in small portions, for example each time new data has arrived at the DLL.

4.3.10.13 NMX_Sampling_ReadRow32_1

Read sampled values row-wise. Row-wise means, that all sampling elements of a single sampling point are read-out. Each time this function is called, the oldest sample data is read-out and deleted afterwards.

See also the [HowTo](#)^[289]-Guide.

Definition

```
NMX_STATUS NMX_Sampling_ReadRow32_1(  
    NMX_PHANDLE pHandle,  
    signed long aslSamples[],  
    unsigned long ulMaxSamples,  
    unsigned long* pulSamplesCopied,  
    unsigned long long* pudSampleNo);
```

Parameter

pHandle

[Connection Handle](#)^[178]

aslSamples

The measurement samples will be written into this array. For simplicity, all sampled values are provided with data type signed long. See chapter "[data types](#)^[164]" for more information.

ulMaxSamples

Maximum number of samples to read-out. This value must be \leq the total number of elements of `aslSamples`.

Example: `aslSamples` is defined as "signed long `aslSamples[64]`". Then `ulMaxSamples` must be ≤ 64 .

pulSamplesCopied

Number of samples copied into `aslSamples`. This value will be \leq `ulMaxSamples`.

pudSampleNo

Number of the this sample, counted from the beginning of sampling, starting at 0.

Example:

The 1. time after the start of sampling, this function is called, `pudSampleNo = 0` will be returned.

The 2. time this function is called, `pudSampleNo = 1` will be returned.

The 3. time this function is called, `pudSampleNo = 2` will be returned.

Typical function call (C example)

Assuming `aslSamples` is defined as: `signed long aslSamples[64];`

```
NMX_Sampling_ReadRow32_1(pHandle, aslSamples, 64,
&ulSamplesCopied, &udNoSample);
```

[.Net DLL](#)¹⁶⁸ specific implementation

```
NMX_MSTATUS Sampling_ReadRow32_1(
    System::IntPtr pHandle,
    array<System::Int32>^aslSamples,
    System::UInt32 %pulSamplesCopied,
    System::UInt64 %pudSampleNo);
```

The array `aslSamples` will not be resized within the function call for performance reasons. This means it should be large enough to store all the data.

If for example 23 sampling elements have been assigned with `NMX_Sampling_Add...`, the the minimum array size should be 23.

If it is smaller, it won't crash but not all data will be available to your application.

Comments

-
- It is possible to get informed about new data by the [notification](#)^[192] NMXNOTIFY_SAMPLING_NEW_DATA.
 - Using NMX_Sampling_ReadRow32_1 it is not possible, reading the same data twice.

4.3.10.14 NMX_Sampling_GetStatus_1

This function is used to read the current sampling status.

See also the [HowTo](#)^[293]-Guide.

Definition

```
NMX_STATUS NMX_Sampling_GetStatus_1(  
    NMX_PHANDLE pHandle,  
    unsigned char* pucStatus,  
    unsigned long* pulNSamplingElements,  
    unsigned long long* pudSamplesReceived,  
    unsigned long long* pudMaxSamples);
```

*Parameter***pHandle**

[Connection Handle](#)^[178]

pucStatus

Current sampling status. See definition below.

pulNSamplingElements

Number of sampling elements used.

pudSamplesReceived

Provide the number of samples, which have been received by the DLL from the device.

pudMaxSamples

Provide the maximum number of samples, that will be recorded.
If you started endless sampling, this value will be 0xFFFFFFFFFFFFFFFF = 18446744073709551615.

Typical function call (C example)

Assuming `aslSamples` is defined as: `signed long aslSamples[64];`

```
NMX_Sampling_ReadRow32_1(pHandle, aslSamples, 64,
&ulSamplesCopied, &udNoSample);
```

Sampling Status

The sampling status is defined as follows:

Status	Hex value	Description
EMS_INACTIVE	0x00	No sampling active.
EMS_PREPARED	0x01	Sampling has been prepared, but not yet started.
EMS_ACTIVE	0x02	Sampling is active.
EMS_DATA_TRANS	0x03	Sampling has ended or has been stopped, but data transfer is still active.
EMS_FINISHED	0x04	Sampling is finished.
EMS_ERROR_CONF	0xF0	Error during configuration / preparation of sampling.
EMS_ERROR_STAR	0xF1	Error during start of sampling.
EMS_ERROR_NOTP	0xF2	Error: start not possible, sampling has not been prepared.
EMS_ERROR_RUN	0xF8	Error occurred while sampling was active.

By definition, all values > 0xF0 are error states.

[.Net DLL](#)^[168] specific implementation

```
NMX_MSTATUS Sampling_GetStatus_1(  
    System::IntPtr pHandle,  
    System::Byte %pucStatus,  
    System::UInt32 %pulNSamplingElements,  
    System::UInt64 %pudSamplesReceived,  
    System::UInt64 %pudMaxSamples);
```

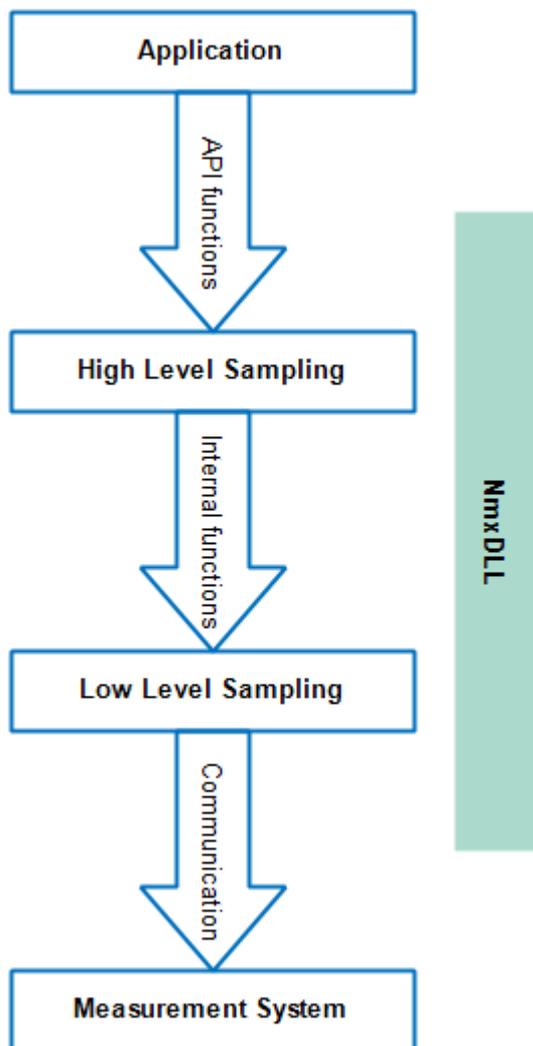
Comments

It is possible to get informed about various sampling events via [notifications](#)^[192].

4.3.11 Sampling HighLevel (Application-specific Realtime Measurement)

Using the high-level functionality is optional! *If only time-triggered realtime measurement is required, using the low-level sampling functionality is sufficient!*

The High-Level Sampling functionality provides an easy way to implement intelligent real-time measurement functions. All high-level functions internally use the [low-level time-triggered realtime sampling](#)^[225].



The following high-level sampling types are available:

- [TFT](#)^[248] stands for "T_{rigger} + F_{ilter} + T_{ail} values" and was developed to meet the specific needs of one customer.

4.3.11.1 NMX_Sampling_PreparePosition_1

Minimum DLL-Version: V1.2.0.13

This function is used to prepare a position-triggered sampling. With position-triggered sampling, one measurement channel is used as a trigger source. In defined position distances, it triggers the other channels. This results in sets of measurement values (samples), which are recorded at equidistant position distances. Technically, position-triggered sampling uses an under

Before starting the sampling, it must be prepared.

See also the [HowTo](#)^[300]**-Guide.**

Definition

```
NMX_STATUS NMX_Sampling_PreparePosition_1(  
    NMX_PHANDLE pHandle,  
    unsigned long ulSamplePeriod,  
    unsigned long ulArrayLength,  
    unsigned long long udMaxSamples,  
    unsigned long ulTriggerChannelNumber,  
    double fdScale,  
    double fdStart,  
    double fdDistance);
```

*Parameter***pHandle**

[Connection Handle](#)^[178]

ulSamplePeriod

Sampling-period for the underlying time-based sampling.
Time period between two samples of the measurement hardware in μs ,
e.g. 1000 for 1ms.
This is the sampling period (-> measurement frequency) which will be
used by the measurement system for acquiring the time-triggered
measurement samples.
[See the chapter "Sampling Speed with Irinos" for more information.](#)^[157]
If you are unsure which value to use, 1000 (= 1ms) is a good value for
starting your development.

ulArrayLength

Triggered data is internally buffered in the DLL. This parameter
specifies the internal buffer length in samples.
Since position-triggered sampling is usually not endless, this parameter
has typically the same value as udMaxSamples.

udMaxSamples

Maximum number of samples, which shall be recorded.
(0 can be used for endless sampling, even though this typically makes
no sense with position-triggered sampling.)

Example: 360° shall be sampled with a position distance of 0.1°. This $udMaxSamples = 360^\circ / 0.1^\circ = 3600$.

To stop position triggering before $udMaxSamples$ has been reached, use [NMX_Sampling_Stop_1](#)^[237].

ulTriggerChannelNumber

Number of the measurement channel, which shall be used as source for the position trigger.

The first channel has the number 0.

Having a device with 24 measurement channels, these are numbered 0..23.

This measurement channel can be part of the sampled channels (this means it has been added to sampling via

[NMX_Sampling_AddChannel_1](#)^[228] or [NMX_Sampling_AddChannelsAll_1](#)^[227]). If this is not the case, it will be added automatically.

fdScale

Format: 64 Bit floating point number according to IEEE 754 (-> double).

Using the scale value, the position value of the Trigger Channel can be converted to a physical unit, e.g. from increments to degrees. It thereby sets the unit of the parameters $fdStart$ and $fdDistance$.

If 1.0 is used, then the raw value from the trigger channel is used.

Example: An encoder with 720000 increments is used. The preferred resolution is 1°. Then $fdScale = 720000 / 360 = 2000$.

If a trigger distance (-> see $fdDistance$) of 0.1° is required, then $fdDistance$ will now be 0.1 instead of 200.

fdStart

Format: 64 Bit floating point number according to IEEE 754 (-> double).

Start position for triggering: Triggering will not start, before this position has been crossed.

In case $fdDistance \geq 0$, the start position must be crossed from a value below $fdStart$ to a value equal/above $fdStart$.

In case $fdDistance \leq 0$, the start position must be crossed from a value above $fdStart$ to a value equal/below $fdStart$.

The unit of this parameter depends on $fdScale$.

fdDistance

Format: 64 Bit floating point number according to IEEE 754 (-> double).

Position distance between two trigger positions.

The unit of this parameter depends on fdScale.

Typical function call (C example)

```
NMX_Sampling_PreparePosition_1(pHandle, 1000, 10000, 10000, 8,  
2000.0f, 0.0f, 0.1f);
```

Comments

-
- For short measurements of a few seconds, it is common practice using the same value for ulArrayLength and for udMaxSamples.
 - The internal buffer is allocated in the heap memory of your application.
 - The larger the DLL-internal buffer is, the larger the memory consumption. For typical time-limited sampling, this is not a problem and the memory size will be between a few kBytes up to a few MBytes. This applies even to larger systems.
However, if you would use for example 256 measurement channels of 32 Bit size, and ulDLLArrayLength would be 100000, then 100 MBytes would be required. This may be too large for your heap memory.

4.3.11.2 NMX_Sampling_PrepareCustomTFT_1

Minimum DLL-Version: V1.1.0.11

This function is used to prepare a time-based sampling with optional triggering, arithmetic average filtering and "recording tail values after stop". Before starting the sampling, it must be prepared.

See also the [HowTo](#)³⁰⁴-Guide.

Definition

```
NMX_STATUS NMX_Sampling_PrepareCustomTFT_1(  
    NMX_PHANDLE pHandle,  
    unsigned long ulSamplePeriod,  
    unsigned long ulFilteredPeriod,  
    unsigned long ulArrayLength,  
    unsigned long long udMaxSamples,
```

```
unsigned long ulNTailSamples,  
unsigned long ulTriggerDigiInByteNo,  
unsigned long ulTriggerDigiInBitNo,  
unsigned long ulTriggerMode,  
unsigned long ulTriggerPolarity);
```

Parameter

pHandle

[Connection Handle](#)^[178]

ulSamplePeriod

Time period between two samples of the measurement hardware in μs , e.g. 1000 for 1ms.

This is the sampling period (-> measurement frequency) which will be used by the measurement system for acquiring the time-triggered measurement samples.

[See the chapter "Sampling Speed with Irinos" for more information.](#)^[157]

ulFilteredPeriod

Time period between two filtered samples provided to the application / measurement software in μs , e.g. 5000 for 5ms.

This value must be an integer multiple of ulSamplePeriod.

With this parameter, an average filter can be applied to the measurement samples. To disable the Filter, use the same values for ulSamplePeriod and ulFilteredPeriod.

Example: ulSamplePeriod is 1ms (=1000), then ulFilteredPeriod can be 1, 2, 3, 4, 5, ..., 10, ... 100ms.

If the SamplePeriod is 1ms and the FilteredPeriod is 5ms, then an average filter of the past 5 samples is used.

ulArrayLength

Data received from the device is internally buffered in the DLL. This parameter specifies the internal buffer length in samples.

Example: The filtered period is 1ms -> 1000Samples/s and the buffer shall contain a maximum of 10 seconds. Then `ulDLLArrayLength = 10000;`

ulMaxSamples

Maximum number of samples, which shall be recorded.

Use 0 for endless sampling.

Example: The sampling speed is 1ms -> 1000Samples/s. Data shall be recorded for a maximum of 15 seconds. Then udMaxSamples = 15000;

ulNTailSamples

If this value is 0, then sampling will be stopped immediately, when the stop condition occurs.

If this value is >0, then ulNTailSamples will be sampled after the stop condition occurs.

The tail samples could for example be used for additional filtering.

ulTriggerDigInByteNo

Number of the digital input byte, which contains the digital input information. Counting starts with 0.

This parameter is only relevant, if Triggering is used (ulTriggerMode > 0).

ulTriggerDigInBitNo

Number of the bit in ulTriggerDigInByteNo, which contains the digital input information. Counting starts with 0, maximum is 7.

This parameter is only relevant, if Triggering is used (ulTriggerMode > 0).

Example: A measurement system has 16 digital inputs, counted from 1 to 16. For input no 11, the parameters are:

ulTriggerDigInByteNo = 1;

ulTriggerDigInBitNo = 2;

ulTriggerMode

See general information about [trigger mode](#)¹⁷⁹.

The following modes are supported:

0 = Trigger disabled

1 = Edge

2 = Level

3 = Edge Start

4 = Level Once

ulTriggerPolarity

1 = falling edge / low active

2 = rising edge / high active

This parameter is only relevant, if Triggering is used (ulTriggerMode > 0).

Typical function call (C example)

```
Time-limited: NMX_Sampling_PrepareCustomTFT_1(pHandle, 1000, 5000, 10000, 10000, 4, 0, 0, 0, 0);
```

```
Endless: NMX_Sampling_PrepareCustomTFT_1(pHandle, 1000, 5000, 10000, 0, 0, 1, 4, 2);
```

Comments

- For short measurements of a few seconds, it is common practice using the same value for ulArrayLength and for udMaxSamples.
- The internal buffer is allocated in the heap memory of your application.
- The larger the DLL-internal buffer is, the larger the memory consumption. For typical time-limited sampling, this is not a problem and the memory size will be between a few kBytes up to a few MBytes. This applies even to larger systems.
However, if you would use for example 256 measurement channels of 32 Bit size, and ulDLLArrayLength would be 100000, then 100 MBytes would be required. This may be too large for your heap memory.

4.3.12 Diagnostics

Your measurement system is equipped with an internal diagnostic service. It supports identifying special events and errors.

4.3.12.1 NMX_DiagClearEvent_1

This function allows clearing an event, which has occurred.

Definition

```
NMX_STATUS NMX_DiagClearEvent_1(  
    NMX_PHANDLE pHandle,  
    unsigned long ulBoxNo,  
    unsigned char ucEventNo);
```

Parameter

pHandle

[Connection Handle](#)^[178]

ulBoxNo

Number of the Box, where the event shall be cleared.

The first measurement box has the number 0.

In a system with 5 measurement boxes, these are numbered 0..4.

ucEventNo

Number of the Event, which shall be cleared.

Current events are readout using [NMX_StaticGet32_1](#)^[214] -> Box-Status.

Typical function call (C example)

```
NMX_DiagClearEvent_1(pHandle, 0, 13);
```

[.Net DLL](#)^[168] specific implementation

```
NMX_MSTATUS DiagClearEvent_1(
    System::IntPtr pHandle,
    System::UInt32 ulBoxNo,
    System::Byte ucEventNo);
```

Comments

Events are always handled Box-wise.

4.3.12.2 NMX_DiagGetEventText_1

This function provides a textual description of an event. It could for example be used to display the event description instead or together with an event number.

Definition

```
NMX_STATUS NMX_DiagGetEventText_1(
    NMX_PHANDLE pHandle,
```

```
unsigned char ucEventNo,  
char* pcText, unsigned long ulSizeofText);
```

Parameter

pHandle

[Connection Handle](#) 

ucEventNo

Number of the event.

pcText

ASCII based string with event description. The maximum string length is 129 characters (128 + Termination).

ulSizeofText

Maximum size of pcText in Bytes/Characters.

Typical function call (C example)

```
NMX_DiagGetEventText_1(pHandle, 15, acText, sizeof(acText));
```

[.Net DLL](#) specific implementation

```
NMX_MSTATUS DiagGetEventText_1(  
    System::IntPtr pHandle,  
    System::Byte ucEventNo,  
    System::String ^%strText);
```

No Length/Sizeof-Parameter is required for the string strText, since this information is not required in a .Net environment. It is directly returned as Unicode-strings. Hence no additional conversion is required.

Comments

4.3.12.3 NMX_SetDateTime_1

This function is used to update the current date & time at the device by using the PC time.

Definition

```
NMX_STATUS NMX_SetDateTime_1(  
    NMX_PHANDLE pHandle);
```

Parameter

pHandle

[Connection Handle](#)¹⁷⁸

Typical function call (C example)

```
NMX_SetDateTime_1(pHandle);
```

[.Net DLL](#)¹⁶⁸ specific implementation

```
NMX_MSTATUS SetDateTime_1(System::IntPtr pHandle);
```

Comments

The measurement system does not have an internal Realtime-Clock. In order to provide advanced information in the diagnostic memory, the current date & time is transferred to the measurement system while connecting.

The internal clock does not take into account leap years of leap seconds. Further it has no high accuracy. Hence it is recommended to rewrite the date/time information once a day. It can be rewritten any time.

The date & time information has no effect on measurement or sampling. It is just for the diagnostic memory.

4.4 HowTo

This chapter is intended as an implementation guideline for the NMX DLL in order to get started quickly.

Another good source for the first steps, is the Demo application, which is available for various programming environments.

4.4.1 Small Measurement Application

For simple applications with low requirements regarding measurement speed, only a small portion of the NMX DLLs functionality is required.

In case reading measurement values from time to time, only 3 simple steps need to be implemented into your application:

1. [Establishing a connection](#)^[255] to the measurement system. This is typically done after the application has been started.
2. [Closing the connection](#)^[257] to the measurement system. This is typically done before the application is closed.
3. [Reading the measurement values cyclically](#)^[259], e.g. within a timer routine. It is good practice to check here for a connection failure. This is done by evaluating the return value of the function [NMX_StaticGet32_1](#)^[214].

Please note: As an alternative to the NMX DLL, a very simple ASCII based interface is available. It can be accessed either via Telnet (similar to RS232) or via UDP. Please consult the respective documentation.

4.4.2 Establishing a connection

A connection is established using [NMX_DeviceIPv4Open_1](#)^[189]. All required connection parameters are assigned via its function parameters. No additional configuration file or similar is required.

Before establishing a connection, a [connection handle](#)^[178] must be declared.

Establishing a connection can last a few hundred milliseconds, since the Windows IP stack may need to determine device-specific network data (e.g. MAC address).

Please note: if a firewall is active, the connection must be allowed by this firewall. With common firewalls, e.g. Windows Firewall, the user is asked one-time, if the connection shall be allowed. This has to be answered with "Yes".

In the following examples, the connection is established to the IP address 192.168.3.99. The port numbers 22517 and 22516 are fixed. It is strongly recommended to store all these parameters in an INI-File, XML-File, in the Registry or something similar.

C/C++

```
// Global definition
NMX_PHANDLE pHandle = NULL;

// Establish the connection.
NMX_STATUS tStatus = NMX_DeviceIPv4Open_1(192, 168, 3, 99,
22517, 22516, &pHandle);
if (tStatus == NST_SUCCESS) {
    // Connection has been established successfully.
    // Do some further initialization here, e.g. register
notifications.
}
else {
    // Failed establishing the connection. Show an error
message.
}
```

Delphi

```
// Global definition
pNmxHandle : NMX_PHANDLE;

// Example: Establish connection within a button event handler.
procedure TMainForm.btnConnectClick(Sender: TObject);
var
    tNmxStat : NMX_STATUS;
begin
    tNmxStat := NMX_DeviceIPv4Open_1(192, 168, 3, 99, 22517, 22516, pNmxHandle);
    if (tNmxStat = NST_SUCCESS) then begin
        // Connection has been established successfully.
        // Do some further initialization here, e.g. register notifications
    end
    else begin
        // Failed establishing the connection. Show an error message.
    end;
end;
```

C# / .Net

```
// Global definition
IntPtr pDevice = IntPtr.Zero;

// Establish the connection.
if (cNmx.DeviceIPv4Open_1(192, 168, 3, 99, 22517, 22516, ref
pDevice) == NMX_MSTATUS.SUCCESS) {
    // Connection has been established successfully.
    // Do some further initialization here, e.g. register
    notifications.
}
else {
    // Failed establishing the connection. Show an error
    message.
}
```

4.4.3 Closing a connection

A connection is closed using [NMX_DeviceClose_1](#)^[191].

Before closing an application, which used the NMX DLL, it must be ensured that all connections are closed. It is common practice to call [NMX_DeviceClose_1](#)^[191] always before closing the measurement application. If no connection is established, nothing happens.

C / C++

```
NMX_DeviceClose_1(&pHandleNmx);
```

Delphi

```
NMX_DeviceClose_1(pNmxHandle);
```

C# / .Net

```
cNmx.DeviceClose_1(ref pDevice);
```

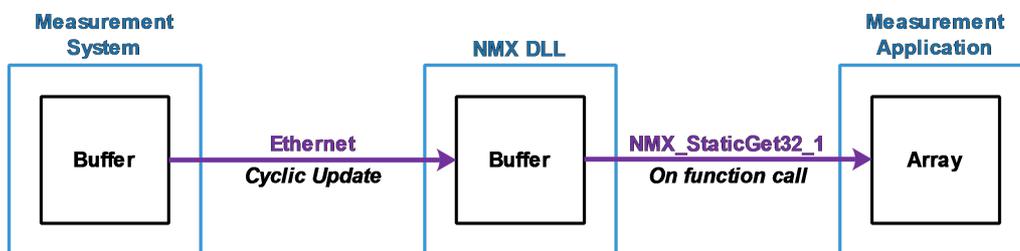
4.4.4 Reading static data

Reading static data is in most cases an essential part of the measurement application. It is very simple to integrate.

The static data is transferred cyclically from the measurement system to the NMX DLL. Here it is stored in the internal buffer.

The typical update rate is approximately 30 Hz, but it is not guaranteed. This cyclic update starts automatically after establishing the connection.

The measurement application can readout the static data from the DLL-internal buffer any time using [NMX_StaticGet32_1](#)^[214]. It will then get the newest data.



Typically all data is read-out via a single function call. There is neither a need nor a possibility to read each measurement channel value individually. The same applied to the other static data.

However, it is important to know, that calling the function [NMX_StaticGet32_1](#)^[214] does not cause any additional data transfer. Thus there is no significant drawback calling it from different places of your source code (but only within the same thread). It is for example no problem reading static measurement values from one part of your software and reading digital input data from another part. Just set the unused pointers to NULL and the respective array size to 0.

There are two common ways of reading static data:

- [Event based](#)^[261] after receiving a notification or
- Calling [NMX_StaticGet32_1](#) [cyclically](#)^[259].

The technically better solution is event based. It is strongly recommended, if [sampling](#)^[266] is used. For very simple applications the cyclic call could be an alternative.

4.4.4.1 Cyclically (Polling)

Cyclically reading data, also known as polling, is the most simple way. It is done by calling [NMX_StaticGet32_1](#)^[214] regularly.

Typically it is called in a time-interval of approximately 30-50 ms.

C/C++

```
// Global definition
#define N_STATIC_CH_DISPLAY      (64) // Max number of
static channels used. Adapt it to your needs.
#define N_DIGIIO_DISPLAY        (128) // Max number of
digital inputs used. Adapt it to your needs.
#define N_MAX_BOXES             (32) // Max number of
boxes supported.

// ###-> Implement timer and timer-event-handler, Interval e.g.
30ms
private: System::Void tmrStatic_Tick(System::Object^ sender,
System::EventArgs^ e) {
    // Disable this timer
    tmrStatic->Enabled = false;

    // Update static data, if new data available.
    signed long aslMeasVal[N_STATIC_CH_DISPLAY];
    unsigned char aucHardStat[N_STATIC_CH_DISPLAY];
    unsigned char aucDigiIn[N_DIGIIO_DISPLAY / 8];
    unsigned char aucBoxStatus[N_MAX_BOXES];
    unsigned long ulNUpdates = 0;
    memset(aslMeasVal, 0, sizeof(aslMeasVal));
    memset(aucHardStat, 0, sizeof(aucHardStat));
    memset(aucDigiIn, 0, sizeof(aucDigiIn));
    memset(aucBoxStatus, 0, sizeof(aucBoxStatus));
    if (NMX_StaticGet32_1(pHandleNmx, aslMeasVal,
N_STATIC_CH_DISPLAY, aucHardStat, sizeof(aucHardStat),
aucDigiIn, sizeof(aucDigiIn), aucBoxStatus,
sizeof(aucBoxStatus), &ulNUpdates) == NST_SUCCESS) {
        // New data successfully received. Display,
Calculate, Store, ...
    }
    else {
        // Error reading data. Typical error:
NST_DX_TIMEOUT_COMMON due to communication breakdown.
    }
}
```

```

    // Re-enable this timer
    tmrStatic->Enabled = true;
}

```

Delphi

```

const cMaxTotalChannels = 256;      // Max number of static channels used. Adapt it to your needs.
      cMaxDigiInBytes = 16;         // Max number of digital input bytes used. Adapt it to your needs.
      cMaxBoxes = 32;              // Max number of boxes supported.

// ###-> Implement timer and timer-event-handler, Interval e.g. 30ms
procedure TMainForm.tmrUpdateStaticTimer(Sender: TObject);
var
  aslValues : array [0..(cMaxTotalChannels-1)] of integer;
  aucHardwareStatus : array [0..(cMaxTotalChannels-1)] of Byte;
  aucDigiInBytes : array [0..(cMaxDigiInBytes-1)] of Byte;
  aucBoxStatus : array [0..(cMaxBoxes-1)] of Byte;
  ulUpdateCtr : Cardinal;
  tStatus : NMX_STATUS;
begin
  tmrUpdateStatic := false;

  tStatus := NMX_StaticGet32_1(pNmxHandle, @aslValues[0], cMaxTotalChannels,
                              @aucHardwareStatus[0], cMaxTotalChannels,
                              @aucDigiInBytes[0], cMaxDigiInBytes,
                              @aucBoxStatus[0], cMaxBoxes,
                              ulUpdateCtr);

  if (tStatus = NST_SUCCESS) then begin
    // New data successfully received. Display, Calculate, Store, ...
  end
  else begin
    // Error reading data. Typical error: NST_DX_TIMEOUT_COMMON due to communication error.
  end;

  tmrUpdateStatic := true;
end;

```

C#/.Net

```

const int N_STATIC_CH_DISPLAY = 64; // Max number of static
channels used. Adapt it to your needs.
const int N_DIGIIO_DISPLAY = 128; // Max number of digital
input bytes used. Adapt it to your needs.
const int N_BOXES_MAX = 32; // Max number of boxes
supported.

// ###-> Implement timer and timer-event-handler, Interval e.g.
30ms

```

```

private void tmrStatic_Tick(object sender, EventArgs e)
{
    /* Disable this timer */
    tmrStatic.Enabled = false;

    Int32[] aslMeasVal = new Int32[N_STATIC_CH_DISPLAY];
    Byte[] aucHardStat = new Byte[N_STATIC_CH_DISPLAY];
    Byte[] aucDigiIn = new Byte[N_DIGIIO_DISPLAY / 8];
    Byte[] aucBoxStatus = new Byte[N_BOXES_MAX];
    UInt32 ulNUpdates = 0;
    if (cNmx.StaticGet32_1(pDevice, aslMeasVal, aucHardStat,
aucDigiIn, aucBoxStatus, ref ulNUpdates) ==
NMX_MSTATUS.SUCCESS)
    {
        // New data successfully received. Display, Calculate,
Store, ...
    }
    else
    {
        // Error reading data. Typical error:
NST_DX_TIMEOUT_COMMON due to communication breakdown.
    }

    /* Re-enable this timer */
    tmrStatic.Enabled = true;
}

```

4.4.4.2 Event based

Reading event based first requires registering a [notification](#)¹⁹². In the following example, message based notifications are used.

A good practice is:

Every-time the notification NMXNOTIFY_NEW_STATIC32 occurs (= a message is received), a flag is set. In a separate timer routine, this flag is checked. If set, static data is readout via [NMX_StaticGet32_1](#)²¹⁴.

If event based data read-out is used, then a connection breakdown is typically handled using the notification NMXNOTIFY_FAILURE_DATA_EXCHANGE. Therefore no special error handling is done in the following sample code.

C/C++

```

// Define Message. In VisualStudio, WM_USER is defined in
WinUser.h as:

```

```

// #define WM_USER 0x0400
#define WM_MESSAGE_NEW_STATIC32 (WM_USER +
NMXNOTIFY_NEW_STATIC32)

// Global declaration
BOOL bNewStatic = false;

// Global definition
#define N_STATIC_CH_DISPLAY (64) // Max number of
static channels used. Adapt it to your needs.
#define N_DIGIIO_DISPLAY (128) // Max number of
digital inputs used. Adapt it to your needs.
#define N_MAX_BOXES (32) // Max number of
boxes supported.

// ###-> Register notification, e.g. directly after connecting.
if (NMX_RegisterMessage_1(pHandle, NMXNOTIFY_NEW_STATIC32,
static_cast<HWND>(Handle.ToPointer()), WM_MESSAGE_NEW_STATIC32,
0, 0) != NST_SUCCESS) {
    // Handle error
}

// ###-> Implement message handler
protected: virtual void WndProc(Message% m) override
{
    /* Listen for operating system messages. */
    switch (m.Msg)
    {
    case WM_MESSAGE_NEW_STATIC32:
        bNewStatic = true;
        break;
    default:
        /* Pass all standard messages to the GUI form. */
        Form::WndProc(m);
        break;
    }
}

// ###-> Implement timer and timer-event-handler, Interval e.g.
30ms
private: System::Void tmrStatic_Tick(System::Object^ sender,
System::EventArgs^ e) {
    // Disable this timer
    tmrStatic->Enabled = false;

    // Update static data, if new data available.
}

```

```

if (bNewStatic != false)
{
    bNewStatic = false;
    signed long aslMeasVal[N_STATIC_CH_DISPLAY];
    unsigned char aucHardStat[N_STATIC_CH_DISPLAY];
    unsigned char aucDigiIn[N_DIGIIO_DISPLAY / 8];
    unsigned char aucBoxStatus[N_MAX_BOXES];
    unsigned long ulNUpdates = 0;
    memset(aslMeasVal, 0, sizeof(aslMeasVal));
    memset(aucHardStat, 0, sizeof(aucHardStat));
    memset(aucDigiIn, 0, sizeof(aucDigiIn));
    memset(aucBoxStatus, 0, sizeof(aucBoxStatus));
    if (NMX_StaticGet32_1(pHandleNmx, aslMeasVal,
N_STATIC_CH_DISPLAY, aucHardStat, sizeof(aucHardStat),
aucDigiIn, sizeof(aucDigiIn), aucBoxStatus,
sizeof(aucBoxStatus), &ulNUpdates) == NST_SUCCESS)
    {
        // New data successfully received. Display,
Calculate, Store, ...
    }
}

// Re-enable this timer
tmrStatic->Enabled = true;
}

```

Delphi

```

// Define Message for "new static data available"
const WM_MESSAGE_NMX_NEWSTATIC = WM_USER + $10;
    cMaxTotalChannels = 256;    // Max number of static channels used. Ad
    cMaxDigiInBytes = 16;      // Max number of digital input bytes used
    cMaxBoxes = 32;           // Max number of boxes supported.

// Declaration of class variables.
bNewStaticData : Boolean;

// Declaration of message handler. Integrate it into the class declaration
procedure OnMessageNewStatic32(var Msg: TMessage); message WM_MESSAGE_NMX_

// ###-> Register notification, e.g. directly after connecting.
NMX_RegisterMessage_1(pNmxHandle, NMXNOTIFY_NEW_STATIC32,
MainForm.Handle, WM_MESSAGE_NMX_NEWSTATIC, 0, 0);

// ###-> Implement message handler
procedure TMainForm.OnMessageNewStatic32(var Msg: TMessage);
begin
    bNewStaticData := true;

```

```

end;

// ###-> Implement timer and timer-event-handler, Interval e.g. 30ms
procedure TMainForm.tmrUpdateStaticTimer(Sender: TObject);
var
  aslValues : array [0..(cMaxTotalChannels-1)] of integer;
  aucHardwareStatus : array [0..(cMaxTotalChannels-1)] of Byte;
  aucDigiInBytes : array [0..(cMaxDigiInBytes-1)] of Byte;
  aucBoxStatus : array [0..(cMaxBoxes-1)] of Byte;
  ulUpdateCtr : Cardinal;
  tStatus : NMX_STATUS;
begin
  tmrUpdateStatic := false;

  if bNewData then begin
    bNewData := false;

    tStatus := NMX_StaticGet32_1(pNmxHandle, @aslValues[0], cMaxTotalChannels,
                                @aucHardwareStatus[0], cMaxTotalChannels,
                                @aucDigiInBytes[0], cMaxDigiInBytes,
                                @aucBoxStatus[0], cMaxBoxes,
                                ulUpdateCtr);

    if (tStatus = NST_SUCCESS) then begin
      // New data successfully received. Display, Calculate, Store, ...
    end;
  end;

  tmrUpdateStatic := true;
end;

```

C#/.Net

```

const int WM_MESSAGE_NEW_STATIC32 = WM_USER + 0x10; // Define
Message for "new static data available"
const int N_STATIC_CH_DISPLAY = 64; // Max number of static
channels used. Adapt it to your needs.
const int N_DIGIIO_DISPLAY = 128; // Max number of digital
input bytes used. Adapt it to your needs.
const int N_BOXES_MAX = 32; // Max number of boxes
supported.

// Global declaration
Boolean bNewStatic = false;

// ###-> Register notification, e.g. directly after connecting.
if (cNmx.RegisterMessage_1(pDevice,
NMX_NOTIFICATION.NEW_STATIC32, Handle, WM_MESSAGE_NEW_STATIC32,
0, 0) != NMX_MSTATUS.SUCCESS)
{
  // Handle error
}

// ###-> Implement message handler

```

```

protected override void WndProc(ref Message m)
{
    // Listen for operating system messages.
    switch (m.Msg)
    {
        case WM_MESSAGE_NEW_STATIC32:
            bNewStatic = true;
            break;

        default:
            /* Pass all standard messages to the GUI form. */
            base.WndProc(ref m);
            break;
    }
}

// ###-> Implement timer and timer-event-handler, Interval e.g.
// 30ms
private void tmrStatic_Tick(object sender, EventArgs e)
{
    /* Disable this timer */
    tmrStatic.Enabled = false;

    if (bNewStatic != false)
    {
        bNewStatic = false;

        Int32[] aslMeasVal = new Int32[N_STATIC_CH_DISPLAY];
        Byte[] aucHardStat = new Byte[N_STATIC_CH_DISPLAY];
        Byte[] aucDigiIn = new Byte[N_DIGIIO_DISPLAY / 8];
        Byte[] aucBoxStatus = new Byte[N_BOXES_MAX];
        UInt32 ulNUpdates = 0;
        if (cNmx.StaticGet32_1(pDevice, aslMeasVal,
            aucHardStat, aucDigiIn, aucBoxStatus, ref ulNUpdates) ==
            NMX_MSTATUS.SUCCESS)
        {
            // New data successfully received. Display,
            Calculate, Store, ...
        }
        else
        {
            // Error reading data. Typical error:
            NST_DX_TIMEOUT_COMMON due to communication breakdown.
        }
    }

    /* Re-enable this timer */
    tmrStatic.Enabled = true;
}

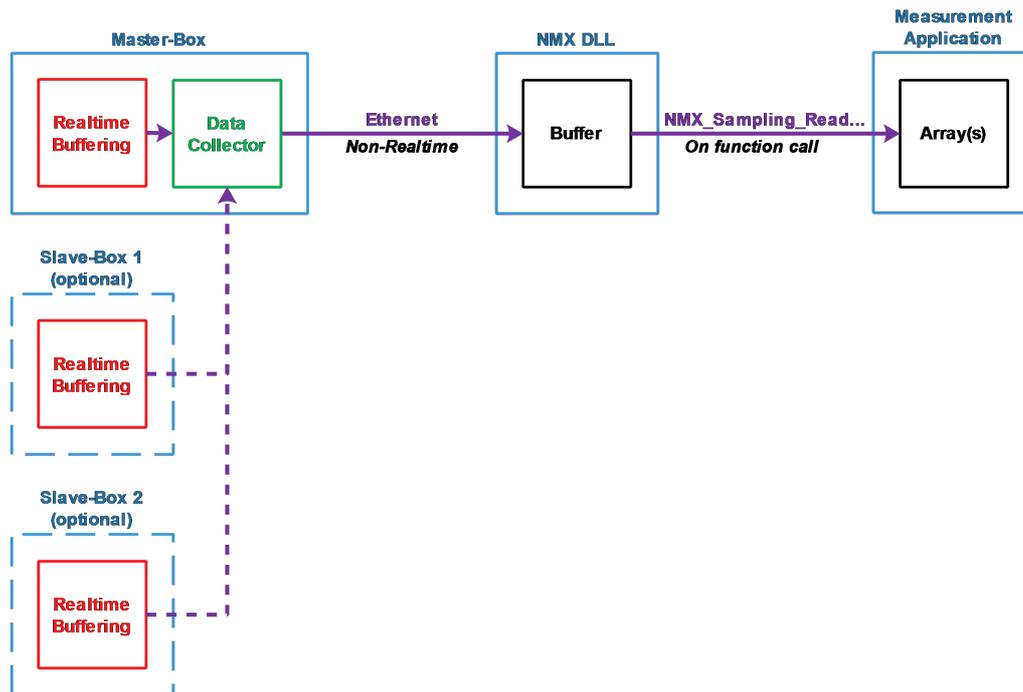
```

4.4.5 Sampling

Using sampling, data is acquired in realtime (see also chapter "[Static vs. Sampling](#)"^[156]). There are two types of sampling:

- Time-limited sampling, and
- Endless sampling.

Both of them use the same buffering technique. The sampled data is first stored in the internal buffer(s) of the measurement system in realtime. For the data transfer to the PC, there are no realtime requirements.



Time-limited sampling provides more opportunities in regard of the measurement speed and the amount of sampled data. Full speed is supported using all measurement channels.

As its name says, it does not run all the time. Instead, it is started before each measurement cycle. It then runs until the measurement cycle is finished. The typical duration is a few seconds.

Endless sampling provides more flexibility. After starting, the sampled data is available endlessly, similar to an endless data stream.

The maximum data rate depends on the amount of sampled data (= amount of measurement channels used). However, for many applications, it is fast enough. Consult the users manual of the measurement system for more information about the maximum sampling rate. Typically "1000 Samples/s

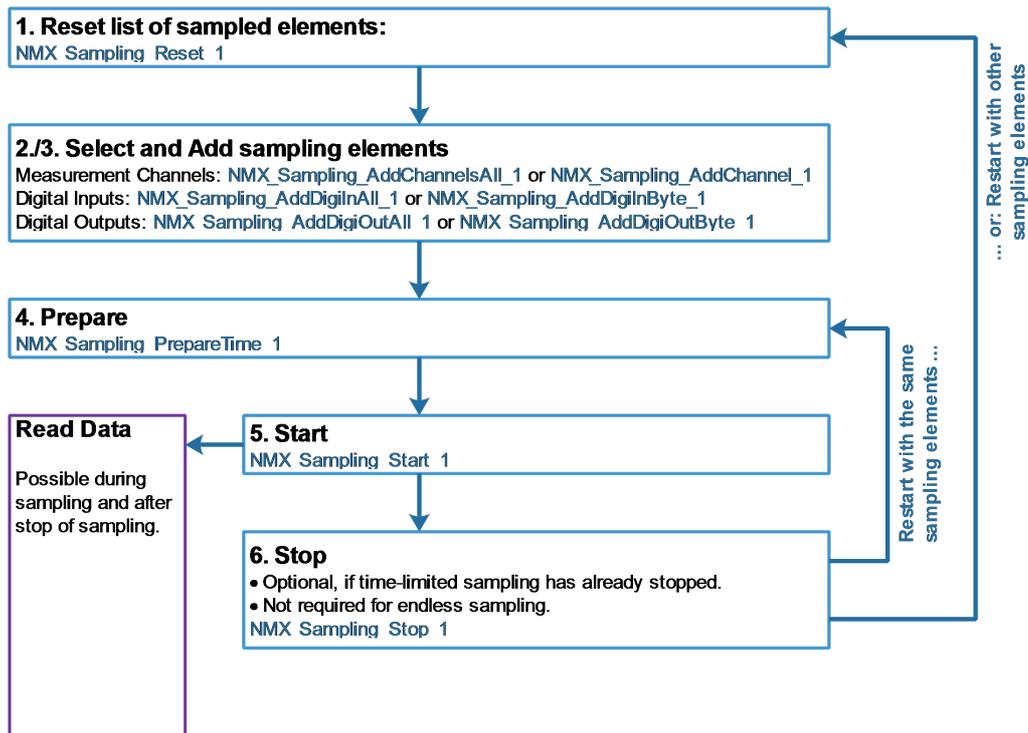
per measurement channel" or more are possible with many measurement channels.

Regarding the integration into the measurement software, both sampling types are similar. The same function calls are used.

Fundamentals are:

- Sampling does not care about measurement boxes. It does not matter whether a single or multiple boxes are used. All measurement channels and digital in-/outputs across the whole measurement system can be used. The boxes are synchronized. The Master-Box collects all data from the Slave-Boxes, sorts it and sends it to the NMX DLL. There it is stored in a table. See the chapter "[Reading sampled data](#)^[283]" for more details.
- At each sampling point, the values of all sampling elements are buffered/stored together with a sample counter.
- As the first step, the sampling elements have to be defined. A sampling element can be a measurement channel, a digital input byte or a digital output byte.
It is possible to add all of them or only a selection.
- The next step is preparing the sampling.
Here the sample rate (-> sample period) is defined. Further it is defined, whether it is a time-limited or an endless sampling.
- Then the sampling can be started.
- The data transfer to the PC begins immediately.

These steps are also shown in the following diagram:



As visible in the diagram, the sampling elements only need to be defined once, if it is started repeatedly.

4.4.5.1 Start endless time-based sampling

For the following sample code, endless sampling with a system having 32 measurement channels and 64 digital in-/outputs is shown.

64 digital in-/outputs means there are 8 bytes each.

The sample rate shall be 1000 Samples/s.

The DLL-internal buffer shall be large enough to buffer for 5 seconds, which equals 5000 samples.

C / C++: Start with all measurement channels and digital I/Os

```

unsigned long ulNElements = 0;

// ###-> 1. Reset list of sampling elements
if (NMX_Sampling_Reset_1(pHandle) == NST_SUCCESS) {
    // List of sampling elements has been reset successfully.
}
else {
    // Failed resetting the list of sampling elements.
    // Do some error handling.
    return;
}
  
```

```
// ###-> 2/3. Add sampling elements
if ( (NMX_Sampling_AddChannelsAll_1(pHandle, &ulNElements) ==
NST_SUCCESS) &&
      (NMX_Sampling_AddDigiInAll_1(pHandle, &ulNElements) ==
NST_SUCCESS) &&
      (NMX_Sampling_AddDigiOutAll_1(pHandle, &ulNElements) ==
NST_SUCCESS) ) {
    // Successfully added all sampling elements
}
else {
    // Failed adding sampling elements
    // Do some error handling.
    return;
}

// ###-> 4. Prepare sampling with 1000 Samples/s (= 1000µs
sample period)
if (NMX_Sampling_PrepareTime_1(pHandle, 1000/*µs*/,
5000/*Buffer-Size*/, 0/*0=NoLimit*/) == NST_SUCCESS) {
    // Sampling successfully prepared
}
else {
    // Failed preparing sampling
    // Do some error handling.
    return;
}

// ###-> 5. Start sampling
if (NMX_Sampling_Start_1(pHandle) == NST_SUCCESS) {
    // Start successful.
    // Reading sampled data should start now/soon to avoid a
buffer overflow.
}
else {
    // Failed starting sampling
    // Do some error handling.
    return;
}
```

C / C++: Start with a selection of measurement channels and digital inputs

```
unsigned long ulNElements = 0;
```

```
// ###-> 1. Reset list of sampling elements
if (NMX_Sampling_Reset_1(pHandle) == NST_SUCCESS) {
    // List of sampling elements has been reset successfully.
}
else {
    // Failed resetting the list of sampling elements.
    // Do some error handling.
    return;
}

// ###-> 2/3. Add measurement channels 0, 5 & 17
//           Add digital input byte 0
if ( (NMX_Sampling_AddChannel_1(pHandle, 0, &ulNElements) ==
NST_SUCCESS) &&
      (NMX_Sampling_AddChannel_1(pHandle, 5, &ulNElements) ==
NST_SUCCESS) &&
      (NMX_Sampling_AddChannel_1(pHandle, 17, &ulNElements) ==
NST_SUCCESS) &&
      (NMX_Sampling_AddDigiInByte_1(pHandle, 0, &ulNElements) ==
NST_SUCCESS) ) {
    // Successfully added selected sampling elements
}
else {
    // Failed adding sampling elements
    // Do some error handling.
    return;
}

// ###-> 4. Prepare sampling with 1000 Samples/s (= 1000µs
sample period)
if (NMX_Sampling_PrepareTime_1(pHandle, 1000/*µs*/,
5000/*Buffer-Size*/, 0/*0=NoLimit*/) == NST_SUCCESS) {
    // Sampling successfully prepared
}
else {
    // Failed preparing sampling
    // Do some error handling.
    return;
}

// ###-> 5. Start sampling
if (NMX_Sampling_Start_1(pHandle) == NST_SUCCESS) {
    // Start successful.
    // Reading sampled data should start now/soon to avoid a
buffer overflow.
}
else {
```

```

    // Failed starting sampling
    // Do some error handling.
    return;
}

```

Delphi: Start with all measurement channels and digital I/Os

```

// ###-> 1. Reset list of sampling elements
if cNmx.Sampling_Reset_1(pHandleNmx) = NST_SUCCESS then begin
    // List of sampling elements has been reset successfully.
end
else begin
    // Failed resetting the list of sampling elements.
    // Do some error handling.
    exit;
end;

// ###-> 2/3. Add sampling elements
if (cNmx.Sampling_AddChannelsAll_1(pHandleNmx, ulNElements) = NST_SUCCESS) And
    (cNmx.Sampling_AddDigiInAll_1(pHandleNmx, ulNElements) = NST_SUCCESS) And
    (cNmx.Sampling_AddDigiOutAll_1(pHandleNmx, ulNElements) = NST_SUCCESS) then begin
    // Successfully added all sampling elements
end
else begin
    // Failed adding sampling elements
    // Do some error handling.
    exit;
end;

// ###-> 4. Prepare sampling with 1000 Samples/s (= 1000µs sample period)
if cNmx.Sampling_PrepareTime_1(pHandleNmx, 1000{µs}, 5000{BufferSize}, 0{0=NoLimit}) = NST_SUCCESS then begin
    // Sampling successfully prepared
end
else begin
    // Failed preparing sampling
    // Do some error handling.
    exit;
end;

// ###-> 5. Start sampling
if cNmx.Sampling_Start_1(pHandleNmx) = NST_SUCCESS then begin
    // Start successful.
    // Reading sampled data should start now/soon to avoid a buffer overflow.
end
else begin
    // Failed starting sampling
    // Do some error handling.
    exit;
end;

```

Delphi: Start with a selection of measurement channels and digital inputs

```

// ###-> 1. Reset list of sampling elements
if cNmx.Sampling_Reset_1(pHandleNmx) = NST_SUCCESS then begin
    // List of sampling elements has been reset successfully.
end
else begin
    // Failed resetting the list of sampling elements.
    // Do some error handling.
    exit;
end;

// ###-> 2/3. Add sampling elements
if (cNmx.Sampling_AddChannel_1(pHandleNmx, 0, ulNTotal) = NST_SUCCESS) And
    (cNmx.Sampling_AddChannel_1(pHandleNmx, 5, ulNTotal) = NST_SUCCESS) And
    (cNmx.Sampling_AddChannel_1(pHandleNmx, 17, ulNTotal) = NST_SUCCESS) And
    (cNmx.Sampling_AddDigiInByte_1(pHandleNmx, 0, ulNTotal) = NST_SUCCESS) then begin
    // Successfully added selected sampling elements
end
else begin
    // Failed adding sampling elements
    // Do some error handling.
    exit;
end;

// ###-> 4. Prepare sampling with 1000 Samples/s (= 1000µs sample period)
if cNmx.Sampling_PrepareTime_1(pHandleNmx, 1000{µs}, 5000{BufferSize}, 0{0=NoLimit}) = NST_SUCCESS then begin
    // Sampling successfully prepared
end
else begin
    // Failed preparing sampling
    // Do some error handling.
    exit;
end;

// ###-> 5. Start sampling
if cNmx.Sampling_Start_1(pHandleNmx) = NST_SUCCESS then begin
    // Start successful.
    // Reading sampled data should start now/soon to avoid a buffer overflow.
end
else begin
    // Failed starting sampling
    // Do some error handling.
    exit;
end;

```

C# / .Net: Start with all measurement channels and digital I/Os

```

UInt32 ulNElements = 0;

```

```
// ###-> 1. Reset list of sampling elements
if (cNmx.Sampling_Reset_1(pDevice) == NMX_MSTATUS.SUCCESS) {
    // List of sampling elements has been reset successfully.
}
else {
    // Failed resetting the list of sampling elements.
    // Do some error handling.
    return;
}

// ###-> 2/3. Add sampling elements
if ( (cNmx.Sampling_AddChannelsAll_1(pDevice, ref ulNElements)
== NMX_MSTATUS.SUCCESS) &&
(cNmx.Sampling_AddDigiInAll_1(pDevice, ref ulNElements) ==
NMX_MSTATUS.SUCCESS) &&
(cNmx.Sampling_AddDigiOutAll_1(pDevice, ref ulNElements)
== NMX_MSTATUS.SUCCESS) ) {
    // Successfully added all sampling elements
}
else {
    // Failed adding sampling elements
    // Do some error handling.
    return;
}

// ###-> 4. Prepare sampling with 1000 Samples/s (= 1000µs
sample period)
if (cNmx.Sampling_PrepareTime_1(pDevice, 1000/*µs*/,
5000/*Buffer-Size*/, 0/*0=NoLimit*/) == NMX_MSTATUS.SUCCESS)
    // Sampling successfully prepared
}
else {
    // Failed preparing sampling
    // Do some error handling.
    return;
}

// ###-> 5. Start sampling
if (cNmx.Sampling_Start_1(pDevice) == NMX_MSTATUS.SUCCESS) {
    // Start successful.
    // Reading sampled data should start now/soon to avoid a
buffer overflow.
}
else {
    // Failed starting sampling
    // Do some error handling.
    return;
}
```

```
}
```

C# / .Net: Start with a selection of measurement channels and digital inputs

```
UInt32 ulNElements = 0;

// ###-> 1. Reset list of sampling elements
if (cNmx.Sampling_Reset_1(pDevice) == NMX_MSTATUS.SUCCESS) {
    // List of sampling elements has been reset successfully.
}
else {
    // Failed resetting the list of sampling elements.
    // Do some error handling.
    return;
}

// ###-> 2/3. Add sampling elements
if ( (cNmx.Sampling_AddChannel_1(pDevice, 0, ref ulNElements)
== NMX_MSTATUS.SUCCESS) &&
(cNmx.Sampling_AddChannel_1(pDevice, 5, ref ulNElements)
== NMX_MSTATUS.SUCCESS) &&
(cNmx.Sampling_AddChannel_1(pDevice, 17, ref ulNElements)
== NMX_MSTATUS.SUCCESS) &&
(cNmx.Sampling_AddDigiInByte_1(pDevice, 0, ref
ulNElements) != NMX_MSTATUS.SUCCESS) ) {
    // Successfully added all sampling elements
}
else {
    // Failed adding sampling elements
    // Do some error handling.
    return;
}

// ###-> 4. Prepare sampling with 1000 Samples/s (= 1000µs
sample period)
if (cNmx.Sampling_PrepareTime_1(pDevice, 1000/*µs*/,
5000/*Buffer-Size*/, 0/*0=NoLimit*/) == NMX_MSTATUS.SUCCESS)
    // Sampling successfully prepared
}
else {
    // Failed preparing sampling
    // Do some error handling.
    return;
}
```

```

// ###-> 5. Start sampling
if (cNmx.Sampling_Start_1(pDevice) == NMX_MSTATUS.SUCCESS) {
    // Start successful.
    // Reading sampled data should start now/soon to avoid a
    buffer overflow.
}
else {
    // Failed starting sampling
    // Do some error handling.
    return;
}

```

4.4.5.2 Start time-limited sampling

For the following sample code, time-limited sampling with a system having 32 measurement channels and 64 digital in-/outputs is shown.

64 digital in-/outputs means there are 8 bytes each.

The sample rate shall be 1000 Samples/s.

The maximum duration shall be 10 seconds, which equals 10000 Samples.
The DLL-internal buffer shall be large enough to store all sampled data.

C / C++: Start with all measurement channels and digital I/Os

```

unsigned long ulNElements = 0;

// ###-> 1. Reset list of sampling elements
if (NMX_Sampling_Reset_1(pHandle) == NST_SUCCESS) {
    // List of sampling elements has been reset successfully.
}
else {
    // Failed resetting the list of sampling elements.
    // Do some error handling.
    return;
}

// ###-> 2/3. Add sampling elements
if ( (NMX_Sampling_AddChannelsAll_1(pHandle, &ulNElements) ==
NST_SUCCESS) &&
    (NMX_Sampling_AddDigiInAll_1(pHandle, &ulNElements) ==
NST_SUCCESS) &&
    (NMX_Sampling_AddDigiOutAll_1(pHandle, &ulNElements) ==
NST_SUCCESS) ) {
    // Successfully added all sampling elements
}

```

```

else {
    // Failed adding sampling elements
    // Do some error handling.
    return;
}

// ###-> 4. Prepare sampling with 1000 Samples/s (= 1000µs
sample period)
if (NMX_Sampling_PrepareTime_1(pHandle, 1000/*µs*/, 10000,
10000/*Samples*/) == NST_SUCCESS) {
    // Sampling successfully prepared
}
else {
    // Failed preparing sampling
    // Do some error handling.
    return;
}

// ###-> 5. Start sampling
if (NMX_Sampling_Start_1(pHandleNmx) == NST_SUCCESS) {
    // Start successful.
}
else {
    // Failed starting sampling
    // Do some error handling.
    return;
}
}

```

C / C++: Start with a selection of measurement channels and digital inputs

```

unsigned long ulNElements = 0;

// ###-> 1. Reset list of sampling elements
if (NMX_Sampling_Reset_1(pHandle) == NST_SUCCESS) {
    // List of sampling elements has been reset successfully.
}
else {
    // Failed resetting the list of sampling elements.
    // Do some error handling.
    return;
}

// ###-> 2/3. Add measurement channels 0, 5 & 17

```

```

//          Add digital input byte 0
if ( (NMX_Sampling_AddChannel_1(pHandle, 0, &lNElements) ==
NST_SUCCESS) &&
      (NMX_Sampling_AddChannel_1(pHandle, 5, &lNElements) ==
NST_SUCCESS) &&
      (NMX_Sampling_AddChannel_1(pHandle, 17, &lNElements) ==
NST_SUCCESS) &&
      (NMX_Sampling_AddDigiInByte_1(pHandle, 0, &lNElements) ==
NST_SUCCESS) ) {
    // Successfully added selected sampling elements
}
else {
    // Failed adding sampling elements
    // Do some error handling.
    return;
}

// ###-> 4. Prepare sampling with 1000 Samples/s (= 1000µs
sample period)
if (NMX_Sampling_PrepareTime_1(pHandle, 1000/*µs*/, 10000,
10000/*Samples*/) == NST_SUCCESS) {
    // Sampling successfully prepared
}
else {
    // Failed preparing sampling
    // Do some error handling.
    return;
}

// ###-> 5. Start sampling
if (NMX_Sampling_Start_1(pHandleNmx) == NST_SUCCESS) {
    // Start successful.
}
else {
    // Failed starting sampling
    // Do some error handling.
    return;
}

```

Delphi: Start with all measurement channels and digital I/Os

```

// ###-> 1. Reset list of sampling elements
if cNmx.Sampling_Reset_1(pHandleNmx) = NST_SUCCESS then begin
    // List of sampling elements has been reset successfully.
end
else begin
    // Failed resetting the list of sampling elements.
    // Do some error handling.

```

```

        exit;
    end;

    // ###-> 2/3. Add sampling elements
    if (cNmx.Sampling_AddChannelsAll_1(pHandleNmx, ulNElements) = NST_SUCCESS) And
        (cNmx.Sampling_AddDigiInAll_1(pHandleNmx, ulNElements) = NST_SUCCESS) And
        (cNmx.Sampling_AddDigiOutAll_1(pHandleNmx, ulNElements) = NST_SUCCESS) then begin
        // Successfully added all sampling elements
    end
    else begin
        // Failed adding sampling elements
        // Do some error handling.
        exit;
    end;

    // ###-> 4. Prepare sampling with 1000 Samples/s (= 1000µs sample period)
    if cNmx.Sampling_PrepareTime_1(pHandleNmx, 1000{µs}, 10000, 10000{Samples}) = NST_S
        // Sampling successfully prepared
    end
    else begin
        // Failed preparing sampling
        // Do some error handling.
        exit;
    end;

    // ###-> 5. Start sampling
    if cNmx.Sampling_Start_1(pHandleNmx) = NST_SUCCESS then begin
        // Start successful.
        // Reading sampled data should start now/soon to avoid a buffer overflow.
    end
    else begin
        // Failed starting sampling
        // Do some error handling.
        exit;
    end;
end;

```

Delphi: Start with a selection of measurement channels and digital inputs

```

// ###-> 1. Reset list of sampling elements
if cNmx.Sampling_Reset_1(pHandleNmx) = NST_SUCCESS then begin
    // List of sampling elements has been reset successfully.
end
else begin
    // Failed resetting the list of sampling elements.
    // Do some error handling.
    exit;
end;

// ###-> 2/3. Add sampling elements
if (cNmx.Sampling_AddChannel_1(pHandleNmx, 0, ulNTotal) = NST_SUCCESS) And

```

```

        (cNmx.Sampling_AddChannel_1(pHandleNmx, 5, ulNTotal) = NST_SUCCESS) And
        (cNmx.Sampling_AddChannel_1(pHandleNmx, 17, ulNTotal) = NST_SUCCESS) And
        (cNmx.Sampling_AddDigiInByte_1(pHandleNmx, 0, ulNTotal) = NST_SUCCESS) then begin
            // Successfully added selected sampling elements
        end
    else begin
        // Failed adding sampling elements
        // Do some error handling.
        exit;
    end;

// ###-> 4. Prepare sampling with 1000 Samples/s (= 1000µs sample period)
if cNmx.Sampling_PrepareTime_1(pHandleNmx, 1000{µs}, 10000, 10000{Samples}) = NST_SUCCESS then begin
    // Sampling successfully prepared
end
else begin
    // Failed preparing sampling
    // Do some error handling.
    exit;
end;

// ###-> 5. Start sampling
if cNmx.Sampling_Start_1(pHandleNmx) = NST_SUCCESS then begin
    // Start successful.
    // Reading sampled data should start now/soon to avoid a buffer overflow.
end
else begin
    // Failed starting sampling
    // Do some error handling.
    exit;
end;
end;

```

C# / .Net: Start with all measurement channels and digital I/Os

```

UInt32 ulNElements = 0;

// ###-> 1. Reset list of sampling elements
if (cNmx.Sampling_Reset_1(pDevice) == NMX_MSTATUS.SUCCESS) {
    // List of sampling elements has been reset successfully.
}
else {
    // Failed resetting the list of sampling elements.
    // Do some error handling.
    return;
}

// ###-> 2/3. Add sampling elements
if ( (cNmx.Sampling_AddChannelsAll_1(pDevice, ref ulNElements)
== NMX_MSTATUS.SUCCESS) &&

```

```

        (cNmx.Sampling_AddDigiInAll_1(pDevice, ref ulNElements) ==
NMX_MSTATUS.SUCCESS) &&
        (cNmx.Sampling_AddDigiOutAll_1(pDevice, ref ulNElements)
== NMX_MSTATUS.SUCCESS) ) {
            // Successfully added all sampling elements
        }
    else {
        // Failed adding sampling elements
        // Do some error handling.
        return;
    }

// ###-> 4. Prepare sampling with 1000 Samples/s (= 1000µs
sample period)
if (cNmx.Sampling_PrepareTime_1(pDevice, 1000/*µs*/, 10000,
10000/*Samples*/) == NMX_MSTATUS.SUCCESS)
    // Sampling successfully prepared
}
else {
    // Failed preparing sampling
    // Do some error handling.
    return;
}

// ###-> 5. Start sampling
if (cNmx.Sampling_Start_1(pDevice) == NMX_MSTATUS.SUCCESS) {
    // Start successful.
    // Reading sampled data should start now/soon to avoid a
buffer overflow.
}
else {
    // Failed starting sampling
    // Do some error handling.
    return;
}

```

C# / .Net: Start with a selection of measurement channels and digital inputs

```

UInt32 ulNElements = 0;

// ###-> 1. Reset list of sampling elements
if (cNmx.Sampling_Reset_1(pDevice) == NMX_MSTATUS.SUCCESS) {
    // List of sampling elements has been reset successfully.
}
else {

```

```
        // Failed resetting the list of sampling elements.
        // Do some error handling.
        return;
    }

    // ###-> 2/3. Add sampling elements
    if ( (cNmx.Sampling_AddChannel_1(pDevice, 0, ref ulNElements)
    == NMX_MSTATUS.SUCCESS) &&
        (cNmx.Sampling_AddChannel_1(pDevice, 5, ref ulNElements)
    == NMX_MSTATUS.SUCCESS) &&
        (cNmx.Sampling_AddChannel_1(pDevice, 17, ref ulNElements)
    == NMX_MSTATUS.SUCCESS) &&
        (cNmx.Sampling_AddDigiInByte_1(pDevice, 0, ref
    ulNElements) != NMX_MSTATUS.SUCCESS) ) {
        // Successfully added all sampling elements
    }
    else {
        // Failed adding sampling elements
        // Do some error handling.
        return;
    }

    // ###-> 4. Prepare sampling with 1000 Samples/s (= 1000µs
    sample period)
    if (cNmx.Sampling_PrepareTime_1(pDevice, 1000/*µs*/, 10000,
    10000/*Samples*/) == NMX_MSTATUS.SUCCESS)
        // Sampling successfully prepared
    }
    else {
        // Failed preparing sampling
        // Do some error handling.
        return;
    }

    // ###-> 5. Start sampling
    if (cNmx.Sampling_Start_1(pDevice) == NMX_MSTATUS.SUCCESS) {
        // Start successful.
        // Reading sampled data should start now/soon to avoid a
        buffer overflow.
    }
    else {
        // Failed starting sampling
        // Do some error handling.
        return;
    }
}
```

4.4.5.3 Stop sampling

Stopping sampling is optional. It can be used to:

- Stopp an endless sampling.
- Stopp a time-limited sampling earlier than defined in the preparation step.
Example: Sampling has been prepared with a length of 10 seconds. After 6.5 seconds the measurement is finished. Sampling shall now be stopped.

Inside the NMX DLL / measurement system, the maximum number of samples is stored during the preparation phase (see [NMX_Sampling_PrepareTime_1](#)^[234]). This value can also be read-out via [NMX_Sampling_GetStatus_1](#)^[242].

By stopping sampling, this value is adjusted to the new end of sampling.

Example: If sampling with 1000 samples/s has been prepared and started for a maximum duration of 10 seconds, the maximum number of samples is 10000. By calling the stop function after 6.5 seconds, this value is adjusted to 6500.

C / C++

```
// ###-> Stop sampling
if (NMX_Sampling_Stop_1(pHandle) == NST_SUCCESS) {
    // Successfully stopped
}
else {
    // Failed stopping sampling
    // Do some error handling.
}
```

Delphi

```
if cNmx.Sampling_Stop_1(pHandleNmx) = NST_SUCCESS then begin
    // Successfully stopped
end
else begin
    // Failed stopping sampling
    // Do some error handling.
end;
```

C# / .Net

```
if (cNmx.Sampling_Stop_1(pDevice) == NMX_MSTATUS.SUCCESS)
```

```

{
    // Successfully stopped
}
else
{
    // Failed stopping sampling
    // Do some error handling.
}

```

4.4.5.4 Reading sampled data

As soon as sampling has been started, data can be read-out.

For time-limited sampling, it is possible to wait until the measurement is finished and then read the data as a single block.

For endless-sampling, data must be read-out cyclically to ensure that the internal buffer of the NMX DLL does not overflow.

Sampled data is provided by the NMX DLL in the form of a table, or technically speaking in the form of a 2-dimensional array. Thereby:

- the rows are the samples
- and the columns are the sampling elements.

The following table provides an example:

Sampling Element	0	1	2	3	4	5
Source	Channel 0	Channel 1	Channel 2	Channel 3	Input Byte 0	Output Byte 0
Sample 0	21722	695	-11256	147236	0x00000003	0x00000000
Sample 1	21725	695	-11260	147924	0x00000003	0x00000000
Sample 2	21729	694	-11266	148626	0x00000003	0x00000002
Sample 3	21733	695	-11279	149301	0x00000003	0x00000002
Sample 4	21734	695	-11288	149998	0x00000003	0x00000002
Sample 5	21736	695	-11298	150654	0x00000004	0x00000002
Sample 6	21743	695	-11305	151131	0x00000004	0x00000004
Sample 7	21749	694	-11326	151840	0x00000004	0x00000004
...
Sample n	26225	695	15318	222516	0x00000008	0x00000000

The rows and columns are indexed as follows:

Sampling Element	0	1	2	3	4	5
Source	Channel 0	Channel 1	Channel 2	Channel 3	Input Byte 0	Output Byte 0
Sample 0	Col 0 Row 0	Col 1 Row 0	Col 2 Row 0	Col 3 Row 0	Col 4 Row 0	Col 5 Row 0
Sample 1	Col 0 Row 1	Col 1 Row 1	Col 2 Row 1	Col 3 Row 1	Col 4 Row 1	Col 5 Row 1
Sample 2	Col 0 Row 2	Col 1 Row 2	Col 2 Row 2	Col 3 Row 2	Col 4 Row 2	Col 5 Row 2
Sample 3	Col 0 Row 3	Col 1 Row 3	Col 2 Row 3	Col 3 Row 3	Col 4 Row 3	Col 5 Row 3
Sample 4	Col 0 Row 4	Col 1 Row 4	Col 2 Row 4	Col 3 Row 4	Col 4 Row 4	Col 5 Row 4
Sample 5	Col 0 Row 5	Col 1 Row 5	Col 2 Row 5	Col 3 Row 5	Col 4 Row 5	Col 5 Row 5
Sample 6	Col 0 Row 6	Col 1 Row 6	Col 2 Row 6	Col 3 Row 6	Col 4 Row 6	Col 5 Row 6
Sample 7	Col 0 Row 7	Col 1 Row 7	Col 2 Row 7	Col 3 Row 7	Col 4 Row 7	Col 5 Row 7
...
Sample n	Col 0 Row n	Col 1 Row n	Col 2 Row n	Col 3 Row n	Col 4 Row n	Col 5 Row n

There are two possibilities for reading this data: [column-wise](#)^[284] and [row-wise](#)^[289]. None of them is generally better:

- Column-wise typically has a better performance, since less function calls are required. However, in most systems/applications, this is not even measurable.
It could be the preferred option for time-limited sampling, if all samples shall be read-out as a whole after the sampling is finished.
- Row-wise could be a little bit easier to use, since there is no need to synchronize the readout of multiple columns. This applies especially to endless sampling.

There is no possibility to read the whole table as a 2-dimensional array, since this is known to be a bad programming style.

Please note that all data is provided as 32 Bit values. For more information please read the chapter "[Data Types](#)^[164]".

4.4.5.4.1 Read Column-Wise

Single block

For time-limited sampling, the easiest way is reading all data together after the measurement is finished. In this case, the function [NMX_Sampling_ReadColumn32_1](#)^[237] must be called one-time per sampling element. The following example shows this visually:

Sampling Element	0	1	2	3	4	5
Source	Channel 0	Channel 1	Channel 2	Channel 3	Input Byte 0	Output Byte 0
Sample 0	21722	695	-11256	147236	0x00000003	0x00000000
Sample 1	21725	695	-11260	147924	0x00000003	0x00000000
Sample 2	21729	694	-11266	148626	0x00000003	0x00000002
Sample 3	21733	695	-11279	149301	0x00000003	0x00000002
Sample 4	21734	695	-11288	149998	0x00000003	0x00000002
Sample 5	21736	695	-11298	150654	0x00000004	0x00000002
Sample 6	21743	695	-11305	151131	0x00000004	0x00000004
Sample 7	21749	694	-11326	151840	0x00000004	0x00000004
Sample 8	21755	695	-11342	152529	0x00000008	0x00000000
Sample 9	21760	695	-11376	153111	0x00000008	0x00000000
Sample 10	1.	2.	3.	4.	0x 5. 08	0x 6. 01
Sample 11					0x 5. 08	0x 6. 01
Sample 12	21824	695	-11467	153722	0x00000008	0x00000001
Sample 13	21823	695	-11468	153710	0x00000008	0x00000001
Sample 14	21824	695	-11468	153717	0x00000008	0x00000001
Sample 15	21824	694	-11467	153720	0x00000008	0x00000001
Sample 16	21824	695	-11467	153709	0x00000008	0x00000001
Sample 17	21823	695	-11467	153716	0x00000008	0x00000001
Sample 18	21824	694	-11467	153722	0x00000008	0x00000001
Sample 19	21824	695	-11467	153719	0x00000008	0x00000001
Sample 20	21824	695	-11466	153711	0x00000008	0x00000001
Sample 21	21824	695	-11466	153716	0x00000008	0x00000001

According to this example, [NMX_Sampling_ReadColumn32_1](#)²³⁷ must be called 6 times. The function/return parameters are as follows:

Function call	ulMaxSamples	ulElementNo	pulSamplesCopied	pudNoFirstSample
1	≥ 22	0	22	0
2	≥ 22	1	22	0
3	≥ 22	2	22	0
4	≥ 22	3	22	0
5	≥ 22	4	22	0
6	≥ 22	5	22	0

The parameter ulDoNotDelete normally is 0. If you need to read all samples again, set it to 1.

Multiple blocks

Another possibility is reading data in small blocks. For endless sampling, this is essential. For time-limited sampling this can be useful, since it allows analysing/displaying data, while sampling is active.

The following example shows visually, how the data from the example above is read in 3 blocks:

Sampling Element	0	1	2	3	4	5
Source	Channel 0	Channel 1	Channel 2	Channel 3	Input Byte 0	Output Byte 0
Sample 0	21722	695	-11256	147236	0x00000003	0x00000000
Sample 1	21725	695	-11260	147234	0x00000003	0x00000000
Sample 2	1	2	3	4	0x 5 03	0x 6 02
Sample 3	21733	695	-11273	149301	0x00000003	0x00000002
Sample 4	21734	695	-11288	149998	0x00000003	0x00000002
Sample 5	21736	695	-11298	150654	0x00000004	0x00000002
Sample 6	21743	695	-11305	151131	0x00000004	0x00000004
Sample 7	21749	694	-11326	151840	0x00000004	0x00000004
Sample 8	7	8	9	10	0x 11 08	0x 12 00
Sample 9					0x 11 08	0x 12 00
Sample 10	21767	694	-11441	153724	0x00000008	0x00000001
Sample 11	21773	695	-11468	153736	0x00000008	0x00000001
Sample 12	21824	695	-11467	153722	0x00000008	0x00000001
Sample 13	21823	695	-11468	153710	0x00000008	0x00000001
Sample 14	21824	695	-11468	153717	0x00000008	0x00000001
Sample 15	21824	694	-11467	153720	0x00000008	0x00000001
Sample 16	21824	695	-11467	153709	0x00000008	0x00000001
Sample 17	13	14	15	16	0x 17 08	0x 18 01
Sample 18					0x 17 08	0x 18 01
Sample 19	21824	695	-11467	153719	0x00000008	0x00000001
Sample 20	21824	695	-11466	153711	0x00000008	0x00000001
Sample 21	21824	695	-11466	153716	0x00000008	0x00000001

According to this example, [NMX_Sampling_ReadColumn32_1](#)²³⁷ must be called 18 times. The function/return parameters are as follows:

Function call	ulMaxSamples	ulElementNo	pulSamplesCopied	pudNoFirstSample
1	5	0	5	0
2	5	1	5	0
3	5	2	5	0
4	5	3	5	0
5	5	4	5	0
6	5	5	5	0
7	9	0	9	5
8	9	1	9	5
9	9	2	9	5
10	9	3	9	5
11	9	4	9	5
12	9	5	9	5
13	≥ 8	0	8	14
14	≥ 8	1	8	14
15	≥ 8	2	8	14
16	≥ 8	3	8	14

17	≥ 8	4	8	14
18	≥ 8	5	8	14

The parameter **ulDoNotDelete** must be 0.

C/C++

```

// ###-> Get Number of Rows and Columns.
unsigned long long udSamplesReceived = 0; // --> Rows
unsigned long ulNSamplingElements = 0;   // --> Columns
if (NMX_Sampling_GetStatus_1(pHandleNmx, NULL,
&ulNSamplingElements, &udSamplesReceived, NULL) != NST_SUCCESS)
{
    // Error reading sampling status. Do some error handling.
    return;
}

// ###-> Create table/array for sampled data
signed long **aas1Samples = new signed
long*[ulNSamplingElements];
unsigned long ulSamplesCopied = 0;

for (unsigned long ulColumn = 0; ulColumn <
ulNSamplingElements; ulColumn++)
{
    // Create array for this column
    aas1Samples[ulColumn] = new signed long[(unsigned long)
udSamplesReceived];
    for (unsigned long ulRow = 0; ulRow < udSamplesReceived;
ulRow++) {
        aas1Samples[ulColumn][ulRow] = -2147483647;
    }

    // ###-> Read sampled data column-wise
    if (NMX_Sampling_ReadColumn32_1(pHandle,
aas1Samples[ulColumn], (unsigned long)udSamplesReceived,
ulColumn, 0, &ulSamplesCopied, NULL) != NST_SUCCESS) {
        // Error reading sampled data. Do some error
handling.
        return;
    }
}

// ###-> Now the data can be processed.
//     First array dimension is columns / sampling elements
//     Second array dimension is rows / samples

```

```
// ###-> Don't forget to delete the array.
delete aas1Samples;
```

Delphi

```
var
  ucStatus: byte;
  udSamplesReceived: uint64;
  udSamplesMax: uint64;
  ulNElements: cardinal;
  aas1Samples: packed array of array of integer;
  ulColumn: cardinal;
  ulSamplesCopied: cardinal;
  udNoFirstSample: uint64;
begin
  ucStatus := 0;
  udSamplesReceived := 0;
  udSamplesMax := 0;
  ulNElements := 0;

  // ###-> Get Number of Rows and Columns.
  if cNmx.Sampling_GetStatus_1(pHandleNmx, ucStatus, ulNElements, udSamplesReceived) < 0 then
    // Error reading sampling status. Do some error handling.
    exit;
end;

// ###-> Create table/array for sampled data
SetLength(aas1Samples, ulNElements);
for ulColumn := 0 to ulNElements-1 do begin
  SetLength(aas1Samples[ulColumn], udSamplesReceived);

  if cNmx.Sampling_ReadColumn32_1(pHandleNmx,
    @aas1Samples[ulColumn][0],
    Length(aas1Samples[0]),
    ulColumn,
    1,
    ulSamplesCopied,
    udNoFirstSample) <> NST_SUCCESS then begin
    // Error reading sampled data. Do some error handling.
  end;
end;

// ###-> Now the data can be processed.
//     First array dimension is columns / sampling elements
//     Second array dimension is rows / samples
end;
```

C#/.Net

```
public struct TSampleColumn
```

```

{
    public Int32[] aslRows;
}

Byte ucStatus = 0;
UInt32 ulNElements = 0;
UInt64 udSamplesReceived = 0;
UInt64 udMaxSamples = 0;
UInt32 ulSamplesCopied = 0;

/* Get number of samples, which are available. */
cNmx.Sampling_GetStatus_1(pDevice, ref ucStatus, ref
ulNElements, ref udSamplesReceived, ref udMaxSamples);

/* Create arrays for sampling data and read it from DLL */
TSampleColumn[] aaslSData = new TSampleColumn[ulNElements];
for (UInt32 ulColumn = 0; ulColumn < ulNElements; ulColumn++)
{
    aaslSData[ulColumn].aslRows = new Int32[udSamplesReceived];
    for (Int32 I = 0; I < aaslSData[ulColumn].aslRows.Length;
I++) {
        aaslSData[ulColumn].aslRows[I] = Int32.MaxValue;
    }

    /* Feel free to do some more error handling here, e.g.
check ulSamplesCopied and checking the function return code. */
    cNmx.Sampling_ReadColumn32_1(pDevice,
        aaslSData[ulColumn].aslRows,
        0,
        (UInt32)
aaslSData[ulColumn].aslRows.Length,
        ulColumn,
        1,
        ref ulSamplesCopied,
        ref udMaxSamples);
}

```

4.4.5.4.2 Read Row-Wise

Reading row-wise is quite simple. The function [NMX_Sampling_ReadRow32_1](#)^[240] is called for each sample and provides the data for all sampling elements. The following example shows this visually.

Sampling Element	0	1	2	3	4	5	
Source	Channel 0	Channel 1	Channel 2	Channel 3	Input Byte 0	Output Byte 0	
Sample 0	21722	695	-11256	147236	0x00000003	0x00000000	1.
Sample 1	21725	695	-11260	147924	0x00000003	0x00000000	2.
Sample 2	21729	694	-11266	148626	0x00000003	0x00000002	3.
Sample 3	21733	695	-11279	149301	0x00000003	0x00000002	4.
Sample 4	21734	695	-11288	149998	0x00000003	0x00000002	5.
Sample 5	21736	695	-11298	150654	0x00000004	0x00000002	6.
Sample 6	21743	695	-11305	151131	0x00000004	0x00000004	7.
Sample 7	21749	694	-11326	151840	0x00000004	0x00000004	8.
Sample 8	21755	695	-11342	152529	0x00000008	0x00000000	9.

According to this example, [NMX_Sampling_ReadRow32_1](#)_[240] must be called 9 times. The function/return parameters are as follows:

Function call	ulMaxSamples	pulSamplesCopied	pudSampleNo
1	≥ 6	6	0
2	≥ 6	6	1
3	≥ 6	6	2
4	≥ 6	6	3
5	≥ 6	6	4
6	≥ 6	6	5
7	≥ 6	6	6
8	≥ 6	6	7
9	≥ 6	6	8

C/C++

```

// ###-> Get Number of Sampling elements. This could be done
// one-time after starting sampling, since this value does not
// change.
unsigned long ulNSamplingElements = 0; // --> Columns
if (NMX_Sampling_GetStatus_1(pHandleNmx, NULL,
&ulNSamplingElements, NULL, NULL) != NST_SUCCESS) {
    // Error reading sampling status. Do some error handling.
    return;
}

// ###-> Read newest data. This code could for example be done
// after the notification NMXNOTIFY_SAMPLING_NEW_DATA.
signed long *aslSamples = new signed long[ulNSamplingElements];
unsigned long ulSamplesCopied = 0;
unsigned long long udSampleNo = 0;
NMX_STATUS tResult = NST_SUCCESS;
    
```

```

do {
    tResult = NMX_Sampling_ReadRow32_1(pHandleNmx,
    aslSamples, ulNSamplingElements, &ulSamplesCopied,
    &udSampleNo);
    if (tResult == NST_SUCCESS) {
        // New row has been read-out. Use the data, e.g.
copy it to your own array or ring-buffer.
    }
    else if (tResult == NST_SAMPLING_NO_DATA_AVAILABLE) {
        // No more data available. Wait until new data has
arrived.
        break;
    }
    else {
        // An error occurred. Do some error handling.
        break;
    }
} while (1);

```

Delphi

```

var
    ucStatus: byte;
    udSamplesReceived: uint64;
    udSamplesMax: uint64;
    ulNElements: cardinal;
    aslSamples: packed array of integer;
    ulSamplesCopied: cardinal;
    udSampleNo: uint64;
    tResult: NMX_STATUS;
begin
    ucStatus := 0;
    udSamplesReceived := 0;
    udSamplesMax := 0;
    ulNElements := 0;

    // ###-> Get Number of Sampling elements. This could be done one-time after start
    if cNmx.Sampling_GetStatus_1(pHandleNmx, ucStatus, ulNElements, udSamplesReceived)
        // Error reading sampling number of sampling elements. Do some error handling
        exit;
    end;

    // ###-> Read newest data. This code could for example be done after the notification
    SetLength(aslSamples, ulNElements);
    while (true) do begin
        tResult := cNmx.Sampling_ReadRow32_1(pHandleNmx, aslSamples, Length(aslSamples));
        if tResult = NST_SUCCESS then begin
            // New row has been read-out. Use the data, e.g. copy it to your own array or
            end
        else if tResult = NST_SAMPLING_NO_DATA_AVAILABLE then begin
            // No more data available. Wait until new data has arrived.
            break;
        end;
    end;

```

```

end
else begin
    // Error reading sampled data. Do some error handling.
    break;
end;
end;
end;
end;

```

C#/.Net

```

Byte ucStatus = 0;
UInt32 ulNElements = 0;
UInt64 udSamplesReceived = 0;
UInt64 udMaxSamples = 0;
UInt32 ulSamplesCopied = 0;

/* Get number of samples, which are available. */
cNmx.Sampling_GetStatus_1(pDevice, ref ucStatus, ref
ulNElements, ref udSamplesReceived, ref udMaxSamples);

/* Create arrays for sampling data and read it from DLL */
Int32[] aslValues = new Int32[ulNElements];

while (true)
{
    for (Int32 I = 0; I < aslValues.Length; I++) aslValues[I] =
Int32.MaxValue;

    /* Read samples from DLL into local array */
    UInt32 ulSamplesCopied = 0;
    NMX_MSTATUS tStatus = cNmx.Sampling_ReadRow32_1(pDevice,
aslValues, ref ulSamplesCopied, ref udNoFirstSample);
    if (tStatus == NMX_MSTATUS.SUCCESS)
    {
        // New row has been read-out. Use the data, e.g. copy
it to your own array or ring-buffer.
    }
    else if (tStatus == NMX_MSTATUS.SAMPLING_NO_DATA_AVAILABLE)
    {
        // No more data available. Wait until new data has
arrived.
        break;
    }
    else
    {
        // An error occurred. Do some error handling.
        break;
    }
}
}

```

4.4.5.5 Get sampling status

After sampling has been started, it is common practice to check its current status. There are two possibilities doing this, whereas it may make sense using both of this in parallel:

- [By polling a function call](#)^[293], the current status and the current sample counter can be read-out from the NMX DLL.
- [Sampling-Notifications](#)^[294] can be used to get informed about certain events.

4.4.5.5.1 Poll sampling status

Polling the sampling status is done by calling the function [NMX_Sampling_GetStatus_1](#)^[242]. The following information is provided:

- Current sampling status (e.g. running, finished, error).
- Number of sampling elements. This is the number of measurement channels + digitale I/O bytes, which have been selected for sampling. After sampling has been prepared, this value remains static.
- The number of samples, which have already been received by the NMX DLL.
- The maximum number of samples, which will be recorded. For endless measurement, this is the maximum value for the respective data type.

C/C++

```

unsigned char ucStatus = 0;
unsigned long ulNElements = 0;
unsigned long long udSamplesReceived = 0;
unsigned long long udSamplesMax = 0;
if (c1Nmx->Sampling_GetStatus_1(pHandleNmx, &ucStatus,
&ulNElements, &udSamplesReceived, &udSamplesMax) ==
NST_SUCCESS) {
    /* Use status data, e.g. display it on the GUI. */
}

```

Delphi

```

procedure TForm1.tmrSamplingTimer(Sender: TObject);
var
    ucStatus: byte;
    udSamplesReceived: uint64;
    udSamplesMax: uint64;

```

```

    ulNElements: cardinal;
begin
    ucStatus := 0;
    udSamplesReceived := 0;
    udSamplesMax := 0;
    ulNElements := 0;

    // Disable this timer
    tmrSampling.Enabled := false;

    if cNmx.Sampling_GetStatus_1(pHandleNmx, ucStatus, ulNElements, udSamplesReceived, udSamplesMax) = 0
        // Use status data, e.g. display it on the GUI.
    end;

    // Re-enable this timer
    tmrSampling.Enabled := true;
end;

```

C#/.Net

```

private void tmrSampling_Tick(object sender, EventArgs e)
{
    /* Disable this timer */
    tmrSampling.Enabled = false;

    Byte ucStatus = 0;
    UInt32 ulNElements = 0;
    UInt64 udSamplesReceived = 0;
    UInt64 udSamplesMax = 0;
    if (cNmx.Sampling_GetStatus_1(pDevice, ref ucStatus, ref ulNElements, ref udSamplesReceived, ref udSamplesMax) ==
        NMX_MSTATUS.SUCCESS)
    {
        /* Use status data, e.g. display it on the GUI. */
    }

    /* Re-enable this timer */
    tmrSampling.Enabled = true;
}

```

4.4.5.5.2 Sampling notifications

Notifications are very useful to get informed about any changes of the sampling. For a complete list of all available notifications, please consult the chapter "[Notifications](#)¹⁹²".

To receive the notifications, these must be registered. Depending on the selected notification type (messages or callbacks), a message handler or a callback function must be implemented.

In the following sample code, message based notifications are used.

C/C++

```
// Define Message. In VisualStudio, WM_USER is defined in
// WinUser.h as:
// #define WM_USER 0x0400
#define WM_MESSAGE_SAMPLING_NEWDATA          (WM_USER +
NMXNOTIFY_SAMPLING_NEW_DATA)                // Message for "new
sampling data" received
#define WM_MESSAGE_SAMPLING_ALLRECEIVED      (WM_USER +
NMXNOTIFY_SAMPLING_ALL_DATA_RECEIVED)       // Message for "all
sampling data has been received"
#define WM_MESSAGE_SAMPLING_FINISHED        (WM_USER +
NMXNOTIFY_SAMPLING_FINISHED)                // Message for
"sampling is finished"
#define WM_MESSAGE_SAMPLING_ERROR           (WM_USER +
NMXNOTIFY_SAMPLING_ERROR)                  // Message for
"sampling error"
#define WM_MESSAGE_SAMPLING_BUFFER_OVERFLOW (WM_USER +
NMXNOTIFY_SAMPLING_BUFFER_OVERFLOW)         // Message for
"sampling buffer overflow in DLL"
#define WM_MESSAGE_SAMPLING_TIMEOUT         (WM_USER +
NMXNOTIFY_SAMPLING_TIMEOUT)                 // Message for
"sampling: timeout receiving data"

// ###-> Register notifications, e.g. directly after
// connecting.
if (NMX_RegisterMessage_1(pHandleNmx,
NMXNOTIFY_SAMPLING_NEW_DATA,
static_cast<HWND>(Handle.ToPointer()),
WM_MESSAGE_SAMPLING_NEWDATA, 0, 0) != NST_SUCCESS) {
    // Handle error
}
if (NMX_RegisterMessage_1(pHandleNmx,
NMXNOTIFY_SAMPLING_ALL_DATA_RECEIVED,
static_cast<HWND>(Handle.ToPointer()),
WM_MESSAGE_SAMPLING_ALLRECEIVED, 0, 0) != NST_SUCCESS) {
    // Handle error
}
if (NMX_RegisterMessage_1(pHandleNmx,
NMXNOTIFY_SAMPLING_FINISHED,
```

```
static_cast<HWND>(Handle.ToPointer()),
WM_MESSAGE_SAMPLING_FINISHED, 0, 0) != NST_SUCCESS) {
    // Handle error
}
if (NMX_RegisterMessage_1(pHandleNmx, NMXNOTIFY_SAMPLING_ERROR,
static_cast<HWND>(Handle.ToPointer()),
WM_MESSAGE_SAMPLING_ERROR, 0, 0) != NST_SUCCESS) {
    // Handle error
}
if (NMX_RegisterMessage_1(pHandleNmx,
NMXNOTIFY_SAMPLING_BUFFER_OVERFLOW,
static_cast<HWND>(Handle.ToPointer()),
WM_MESSAGE_SAMPLING_BUFFER_OVERFLOW, 0, 0) != NST_SUCCESS) {
    // Handle error
}
if (NMX_RegisterMessage_1(pHandleNmx,
NMXNOTIFY_SAMPLING_TIMEOUT,
static_cast<HWND>(Handle.ToPointer()),
WM_MESSAGE_SAMPLING_TIMEOUT, 0, 0) != NST_SUCCESS) {
    // Handle error
}

// ###-> Implement message handler
// Please note: Keep the code in the message handler as short
as possible.
// Do not update the GUI from within the message handlers.
protected: virtual void WndProc(Message% m) override
{
    /* Listen for operating system messages. */
    switch (m.Msg)
    {
        case WM_MESSAGE_SAMPLING_NEWDATA:
            break;

        case WM_MESSAGE_SAMPLING_ALLRECEIVED:
            break;

        case WM_MESSAGE_SAMPLING_FINISHED:
            break;

        case WM_MESSAGE_SAMPLING_ERROR:
            break;

        case WM_MESSAGE_SAMPLING_BUFFER_OVERFLOW:
            break;

        case WM_MESSAGE_SAMPLING_TIMEOUT:
            break;
    }
}
```

```

    default:
        /* Pass all standard messages to the GUI form. */
        Form::WndProc(m);
        break;
    }
}

```

Delphi

```

// Define Message for "new static data available"
const WM_MESSAGE_NMX_SAMPLING_NEW_DATA      = WM_USER + NMXNOTIFY_SAMPLING_NEW_DATA
      WM_MESSAGE_NMX_SAMPLING_FINISHED      = WM_USER + NMXNOTIFY_SAMPLING_FINISHED
      WM_MESSAGE_NMX_SAMPLING_ALL_DATA_RECEIVED = WM_USER + NMXNOTIFY_SAMPLING_ALL_DATA_RECEIVED
      WM_MESSAGE_NMX_SAMPLING_ERROR        = WM_USER + NMXNOTIFY_SAMPLING_ERROR
      WM_MESSAGE_NMX_SAMPLING_BUFFER_OVERFLOW = WM_USER + NMXNOTIFY_SAMPLING_BUFFER_OVERFLOW
      WM_MESSAGE_NMX_SAMPLING_TIMEOUT      = WM_USER + NMXNOTIFY_SAMPLING_TIMEOUT

// Declaration of message handler. Integrate it into the class declaration
procedure OnMessageSamplingNewData(var Msg: TMessage); message WM_MESSAGE_NMX_SAMPLING_NEW_DATA;
procedure OnMessageSamplingFinished(var Msg: TMessage); message WM_MESSAGE_NMX_SAMPLING_FINISHED;
procedure OnMessageSamplingAllDataReceived(var Msg: TMessage); message WM_MESSAGE_NMX_SAMPLING_ALL_DATA_RECEIVED;
procedure OnMessageSamplingError(var Msg: TMessage); message WM_MESSAGE_NMX_SAMPLING_ERROR;
procedure OnMessageSamplingBufferOverflow(var Msg: TMessage); message WM_MESSAGE_NMX_SAMPLING_BUFFER_OVERFLOW;
procedure OnMessageSamplingTimeout(var Msg: TMessage); message WM_MESSAGE_NMX_SAMPLING_TIMEOUT;

// ###-> Register notification, e.g. directly after connecting.
if NMX_RegisterMessage_1(pNmxHandle,
  NMXNOTIFY_SAMPLING_NEW_DATA, MainForm.Handle,
  WM_MESSAGE_NMX_SAMPLING_NEW_DATA, 0, 0) <> NST_SUCCESS then
begin
    // Registering notification failed. Do some error handling
end;
if NMX_RegisterMessage_1(pNmxHandle,
  NMXNOTIFY_SAMPLING_FINISHED, MainForm.Handle,
  WM_MESSAGE_NMX_SAMPLING_FINISHED, 0, 0) <> NST_SUCCESS then
begin
    // Registering notification failed. Do some error handling
end;
if NMX_RegisterMessage_1(pNmxHandle,
  NMXNOTIFY_SAMPLING_ALL_DATA_RECEIVED, MainForm.Handle,
  WM_MESSAGE_NMX_SAMPLING_ALL_DATA_RECEIVED, 0, 0) <>
  NST_SUCCESS then begin
    // Registering notification failed. Do some error handling
end;
if NMX_RegisterMessage_1(pNmxHandle,
  NMXNOTIFY_SAMPLING_ERROR, MainForm.Handle,
  WM_MESSAGE_NMX_SAMPLING_ERROR, 0, 0) <> NST_SUCCESS then
begin
    // Registering notification failed. Do some error handling
end;
if NMX_RegisterMessage_1(pNmxHandle,
  NMXNOTIFY_SAMPLING_BUFFER_OVERFLOW, MainForm.Handle,

```

```

WM_MESSAGE_NMX_SAMPLING_BUFFER_OVERFLOW, 0, 0) <>
NST_SUCCESS then begin
    // Registering notification failed. Do some error handling
end;
if NMX_RegisterMessage_1(pNmxHandle,
NMXNOTIFY_SAMPLING_TIMEOUT, MainForm.Handle,
WM_MESSAGE_NMX_SAMPLING_TIMEOUT, 0, 0) <> NST_SUCCESS then
begin
    // Registering notification failed. Do some error handling
end;

// ###-> Implement message handlers
// Please note: Keep the code in the message handlers as short as possible
// Do not update the GUI from within the message handlers.

```

C# /.Net

```

// Define Messages. In VisualStudio, WM_USER is 0x0400
const int WM_USER = 0x400;
const int WM_MESSAGE_SAMPLING_NEW_DATA = WM_USER + 0x20;
const int WM_MESSAGE_SAMPLING_FINISHED = WM_USER + 0x21;
public const int WM_MESSAGE_SAMPLING_ALL_DATA_RECEIVED =
WM_USER + 0x22;
public const int WM_MESSAGE_SAMPLING_ERROR = WM_USER + 0x28;
public const int WM_MESSAGE_SAMPLING_BUFFER_OVERFLOW = WM_USER
+ 0x29;
public const int WM_MESSAGE_SAMPLING_TIMEOUT = WM_USER + 0x2A;

// ###-> Register notifications, e.g. directly after
connecting.
NMX_MSTATUS tStatus = NMX_MSTATUS.UNKNOWN;
tStatus = cNmx.RegisterMessage_1(pDevice,
NMX_NOTIFICATION.SAMPLING_NEW_DATA, Handle,
WM_MESSAGE_SAMPLING_NEW_DATA, 0, 0);
if (tStatus != NMX_MSTATUS.SUCCESS) { /* Do some error handling
*/ }
tStatus = cNmx.RegisterMessage_1(pDevice,
NMX_NOTIFICATION.SAMPLING_ALL_DATA_RECEIVED, Handle,
WM_MESSAGE_SAMPLING_ALL_DATA_RECEIVED, 0, 0);
if (tStatus != NMX_MSTATUS.SUCCESS) { /* Do some error handling
*/ }
tStatus = cNmx.RegisterMessage_1(pDevice,
NMX_NOTIFICATION.SAMPLING_FINISHED, Handle,
WM_MESSAGE_SAMPLING_FINISHED, 0, 0);
if (tStatus != NMX_MSTATUS.SUCCESS) { /* Do some error handling
*/ }
tStatus = cNmx.RegisterMessage_1(pDevice,
NMX_NOTIFICATION.SAMPLING_ERROR, Handle,
WM_MESSAGE_SAMPLING_ERROR, 0, 0);

```

```
if (tStatus != NMX_MSTATUS.SUCCESS) { /* Do some error handling
*/ }
tStatus = cNmx.RegisterMessage_1(pDevice,
NMX_NOTIFICATION.SAMPLING_BUFFER_OVERFLOW, Handle,
WM_MESSAGE_SAMPLING_BUFFER_OVERFLOW, 0, 0);
if (tStatus != NMX_MSTATUS.SUCCESS) { /* Do some error handling
*/ }
tStatus = cNmx.RegisterMessage_1(pDevice,
NMX_NOTIFICATION.SAMPLING_TIMEOUT, Handle,
WM_MESSAGE_SAMPLING_TIMEOUT, 0, 0);
if (tStatus != NMX_MSTATUS.SUCCESS) { /* Do some error handling
*/ }

// ###-> Implement message handler
// Please note: Keep the code in the message handler as short
as possible.
// Do not update the GUI from within the message handlers.
protected override void WndProc(ref Message m)
{
    // Listen for operating system messages.
    switch (m.Msg)
    {
        case WM_MESSAGE_SAMPLING_NEW_DATA:
            break;

        case WM_MESSAGE_SAMPLING_FINISHED:
            break;

        case WM_MESSAGE_SAMPLING_ALL_DATA_RECEIVED:
            break;

        case WM_MESSAGE_SAMPLING_ERROR:
            break;

        case WM_MESSAGE_SAMPLING_BUFFER_OVERFLOW:
            break;

        case WM_MESSAGE_SAMPLING_TIMEOUT:
            break;

        default:
            /* Pass all standard messages to the GUI form. */
            base.WndProc(ref m);
            break;
    }
}
```

4.4.5.6 Start position triggered sampling

Instead of recording the measured values at constant time intervals, the position-triggered measurement enables the acquisition at constant position intervals, e.g. in 0.1° or $10\mu\text{m}$ distances.

If position-triggered sampling is started, the NMX DLL internally starts an endless time-triggered sampling via the low-level sampling functions. With the Parameter `ulSamplingSpeed`, the speed of this sampling is defined. Position triggered sampling then reads all the sampled data and processes it. The processed data is then read by the application / measurement software. The accuracy of the position distance depends on the time interval used for sampling.

This is important to know, since the underlying endless time-triggered sampling will only work endless, as long as the sampled data can be transferred from the measurement system to the PC in realtime. For many applications, $1000 \text{ samples/s} = 1000 \mu\text{s}$ should be enough and these can almost always be transferred in realtime. In case a higher speed is required, please consult the chapter "[Sampling Speed with Irinos^{\[157\]}](#)" for a more detailed information.

Setting up the trigger

For position triggered sampling, it is required to provide the NMX DLL information about the trigger. That information is:

1. Which measurement channel does provide the position information? -> Which encoder or probe does provide the position information, which is required to identify the trigger points?
In the function call of [NMX_Sampling_PreparePosition_1^{\[245\]}](#), this is the parameter `ulTriggerChannelNumber`.
2. Where / at which position shall triggering be started?
In the function call of [NMX_Sampling_PreparePosition_1^{\[245\]}](#), this is the parameter `fdStart`.
3. What is the position distance between two trigger points?
In the function call of [NMX_Sampling_PreparePosition_1^{\[245\]}](#), this is the parameter `fdDistance`.

Optionally, the unit for the parameters `fdStart` and `fdDistance` can be set via a scale factor. In the function call of [NMX_Sampling_PreparePosition_1^{\[245\]}](#), this is the parameter `fdScale`. If `fdScale = 1`, then no scaling is used.

With these parameters, sometimes several possibilities lead to the same goal. Thus not every possibility for setting up the trigger can be discussed here. However, following two examples are provided:

Example 1:

A measurement system consists of 8 inductive + 4 incremental measurement channels. A rotational incremental encoder is connected to the 2nd incremental channel. The encoder has a resolution of 400000 increments per revolution. Measurement shall be started at 0° and end after one rotation with a position distance of 0.5°. Then the following parameters could be used:

```
ulTriggerChannelNumber = 9;      // 10-1 = 9, since channel numbering is
0-based
fdScale = 400000.0 / 360.0;      // = 1111.1111
fdStart = 0.0;
fdDistance = 0.5;
udMaxSamples = 360.0 / 0.5;      // = 720.0
```

Example 2:

A measurement system consists of 4 incremental measurement channels + 32 inductive measurement channels. A linear encoder is connected to the 1st incremental channel. Measurement shall be started at the position 7500 increments towards the negative direction with a position distance of 100 increments. Measurement shall be stopped at -423600 Then the following parameters could be used:

```
ulTriggerChannelNumber = 0;      // 1-1 = 0, since channel numbering is 0-
based
fdScale = 1.0;                   // No scale factor is used
fdStart = 7500.0;
fdDistance = -100.0;             // Minus due to negative direction
udMaxSamples = 4312;            // = ((7500 - (-423600)) / 100) + 1
```

Coding position-triggered sampling

Basically, position-triggered sampling is used in the same way as standard low-level sampling. Therefore most of the [low-level functions](#)^[225] can be used:

- [NMX_Sampling_GetMaxSpeed_1](#)^[225]
- [NMX_Sampling_Reset_1](#)^[226]
- [NMX_Sampling_AddChannelsAll_1](#)^[227]
- [NMX_Sampling_AddChannel_1](#)^[228]
- [NMX_Sampling_AddDigiInAll_1](#)^[230]
- [NMX_Sampling_AddDigiInByte_1](#)^[231]
- [NMX_Sampling_AddDigiOutAll_1](#)^[232]
- [NMX_Sampling_AddDigiOutByte_1](#)^[233]
- [NMX_Sampling_Start_1](#)^[236]
- [NMX_Sampling_Stop_1](#)^[237]
- [NMX_Sampling_ReadColumn32_1](#)^[237]
- [NMX_Sampling_ReadRow32_1](#)^[240]
- [NMX_Sampling_GetStatus_1](#)^[242]

The major difference is that the function [NMX_Sampling_PreparePosition_1](#)^[245] instead of the function [NMX_Sampling_PrepareTime_1](#)^[234] is used.

The use of [notifications](#)^[192] is the same as with low-level sampling.

As a result of this, the following other HowTo's can be used as well:

- [Start endless time-based sampling](#)^[268], except that [NMX_Sampling_PrepareCustomTFT_1](#)^[248] must be used.
- [Start time-limited sampling](#)^[275], except that [NMX_Sampling_PrepareCustomTFT_1](#)^[248] must be used.
- [Stop sampling](#)^[282]
- [Reading sampled data](#)^[283]
- [Get sampling status](#)^[293]

Following one example is provided for starting position-triggered sampling:

C / C++: Start with all measurement channels, 250µs sample period, first encoder as trigger source, scale factor 200.0, start position 0°, distance 0.5° and stop after one rotation

```
unsigned long ulNElements = 0;

// ###-> 1. Reset list of sampling elements
if (NMX_Sampling_Reset_1(pHandle) == NST_SUCCESS) {
    // List of sampling elements has been reset successfully.
}
else {
    // Failed resetting the list of sampling elements.
    // Do some error handling.
    return;
}

// ###-> 2/3. Add sampling elements
if (NMX_Sampling_AddChannelsAll_1(pHandle, &ulNElements) ==
NST_SUCCESS) {
    // Successfully added all sampling elements
}
else {
    // Failed adding sampling elements
    // Do some error handling.
    return;
}

NMX_STATUS NMX_Sampling_PreparePosition_1(
    NMX_PHANDLE pHandle,
    unsigned long ulSamplePeriod,
    unsigned long ulArrayLength,
    unsigned long long udMaxSamples,
    unsigned long ulTriggerChannelNumber,
    double fdScale,
    double fdStart,
    double fdDistance);

// ###-> 4. Prepare sampling
if (NMX_Sampling_PreparePosition_1(pHandle, 250/*µs*/, 720,
720/*MaxSamples*/, 0 /*first channel*/, 200.0, 0.0, 0.5) ==
NST_SUCCESS) {
    // Sampling successfully prepared
}
else {
    // Failed preparing sampling
```

```
        // Do some error handling.
        return;
    }

    // ###-> 5. Start sampling
    if (NMX_Sampling_Start_1(pHandleNmx) == NST_SUCCESS) {
        // Start successful.
    }
    else {
        // Failed starting sampling
        // Do some error handling.
        return;
    }
}
```

4.4.5.7 Start TFT high-level sampling

TFT sampling is similar to time-limited sampling, but provides 3 features. These are

- triggering by a digital input,
- applying an arithmetic average filter on the measurement values and
- providing additional samples, called "tail samples", after the sampling has been stopped.

Each of them can be disabled individually.

If TFT sampling is started, the NMX DLL internally starts an endless time-triggered sampling via the low-level sampling functions. With the Parameter `ulSamplingSpeed`, the speed of this sampling is defined. TFT sampling then reads all the sampled data and processes it. The processed data is then read by the application / measurement software.

This is important to know, since the underlying endless time-triggered sampling will only work endless, as long as the sampled data can be transferred from the measurement system to the PC in realtime. For many applications, 1000 samples/s = 1000 μ s should be enough and these can almost always be transferred in realtime. In case a higher speed is required, please consult the chapter "[Sampling Speed with Irinos](#)¹⁵⁷" for a more detailed information.

Triggering

TFT sampling provides several possibilities to start / stop the recording of measurement values via a digital input. The trigger modes "Edge", "Level",

"Edge Start" and "Level Once" are available. For more information about these, consult the chapter "[Trigger Modes](#)^[179]".

Here are further notes:

- Triggering can be disabled completely by using the dummy trigger mode "Off".
- Sampling can always be stopped manually by calling [NMX_Sampling_Stop_1](#)^[237]. The digital input then has no effect.
- Besides using a digital input for triggering, all digital input data can still be included into the list of sampling elements (see [NMX_Sampling_AddDigInAll_1](#)^[230] and [NMX_Sampling_AddDigInByte_1](#)^[231]).

Filtering

Via an integrated arithmetic average filter, the measurement values can be smoothened. The results are provided to the user application / measurement software.

Setting the filter is done via the parameters `ulSamplePeriod` and `ulFilterPeriod`:

- If `ulSamplePeriod` = `ulFilterPeriod`, then filtering is disabled.
- `ulFilterPeriod` must be an integer multiple of `ulSamplePeriod`.
Example for `ulSamplePeriod` = 1000 (=1 ms):
Valid values for `ulFilterPeriod` are 1000, 2000, 3000, 4000, 5000, ..., 10000.
But for example 2500 would be invalid.
- `ulFilterPeriod` must be \geq `ulSamplePeriod`

Example:

`ulSamplePeriod` = 1000, which is 1ms or 1000 samples/s.

`ulFilterPeriod` = 5000, which is 5ms or 200 samples/s.

-> The arithmetic average of 5 incoming samples is calculated and provided to the application / measurement software. This means the application receives 200 samples/s.

Note: Digital inputs or outputs are never filtered.

Tail values

Tail values are recorded after stop of sampling, no matter how the stop condition occurred. Typically they are used to apply an additional filter in the application / measurement software.

The parameter ulNTailSamples defines the number of samples recorded after stop. If this value is 0, then recording tail values is disabled.

Tail values can also be used with endless sampling. After endless sampling is stopped manually via [NMX_Sampling_Stop_1](#)^[237], the tail values will be recorded.

Coding TFT high-level sampling

Basically, TFT high-level sampling is used in the same way as standard low-level sampling. Therefore most of the [low-level functions](#)^[225] can be used:

- [NMX_Sampling_GetMaxSpeed_1](#)^[225]
- [NMX_Sampling_Reset_1](#)^[226]
- [NMX_Sampling_AddChannelsAll_1](#)^[227]
- [NMX_Sampling_AddChannel_1](#)^[228]
- [NMX_Sampling_AddDigiInAll_1](#)^[230]
- [NMX_Sampling_AddDigiInByte_1](#)^[231]
- [NMX_Sampling_AddDigiOutAll_1](#)^[232]
- [NMX_Sampling_AddDigiOutByte_1](#)^[233]
- [NMX_Sampling_Start_1](#)^[236]
- [NMX_Sampling_Stop_1](#)^[237]
- [NMX_Sampling_ReadColumn32_1](#)^[237]
- [NMX_Sampling_ReadRow32_1](#)^[240]
- [NMX_Sampling_GetStatus_1](#)^[242]

The major difference is that the function [NMX_Sampling_PrepareCustomTFT_1](#)^[248] instead of the function [NMX_Sampling_PrepareTime_1](#)^[234] is used.

The use of [notifications](#)^[192] is the same as with low-level sampling.

As a result of this, the following other HowTo's can be used as well:

- [Start endless time-based sampling](#)^[268], except that [NMX_Sampling_PrepareCustomTFT_1](#)^[248] must be used.
- [Start time-limited sampling](#)^[275], except that [NMX_Sampling_PrepareCustomTFT_1](#)^[248] must be used.
- [Stop sampling](#)^[282]
- [Reading sampled data](#)^[283]
- [Get sampling status](#)^[293]

Following one example is provided for starting TFT high-level sampling:

C / C++: Start with all measurement channels, 1ms sample period, 5ms filter period, trigger "Level Once" and 4 Tail values

```

unsigned long ulNElements = 0;

// ###-> 1. Reset list of sampling elements
if (NMX_Sampling_Reset_1(pHandle) == NST_SUCCESS) {
    // List of sampling elements has been reset successfully.
}
else {
    // Failed resetting the list of sampling elements.
    // Do some error handling.
    return;
}

// ###-> 2/3. Add sampling elements
if (NMX_Sampling_AddChannelsAll_1(pHandle, &ulNElements) ==
NST_SUCCESS) {
    // Successfully added all sampling elements
}
else {
    // Failed adding sampling elements
    // Do some error handling.
    return;
}

// ###-> 4. Prepare sampling
if (NMX_Sampling_PrepareCustomTFT_1(pHandle, 1000/*µs*/,
5000/*µs*/, 10000, 10000/*MaxSamples*/, 4, 0, 3, 4, 2) ==
NST_SUCCESS) {
    // Sampling successfully prepared
}

```

```
else {
    // Failed preparing sampling
    // Do some error handling.
    return;
}

// ###-> 5. Start sampling
if (NMX_Sampling_Start_1(pHandleNmx) == NST_SUCCESS) {
    // Start successful.
}
else {
    // Failed starting sampling
    // Do some error handling.
    return;
}
```

- 1 -

1Vss 94, 121

- A -

Absolutzeit 125
Auto-MDI(X) 94, 106

- B -

Box 199, 200

- C -

Channel 182, 186, 204, 205, 206, 227, 228
Connection 178, 189, 191, 192, 255, 257

- D -

Default Gateway 107, 115
DHCP 94, 100, 110
Diagnose-Speicher 130
Digital I/Os 211, 213, 221, 222, 230, 231, 232, 233
DLL 94
DNS 94
Dynamische Messung 128

- E -

Ethernet 100

- F -

Firmware-Update 132

- G -

Gateway 115

- I -

ILink 94
Inventar 123
IP 94
IP-Adresse 107, 112, 114, 115
IP-Konfiguration 115

- L -

LAN 94

- M -

MAC 94
MAC-Adresse 114
MDI 94
MDI-X 94
Msc.cfg 103
MscDII 103

- P -

PC 94

- R -

RS422 121

- S -

Sicherheitshinweise 98
Statische Messung 127
Subnetzmaske 107, 112, 114, 115

- T -

TCP 94
TTL 94, 121

- U -

UDP 94

- V -

Versionsnummer 131