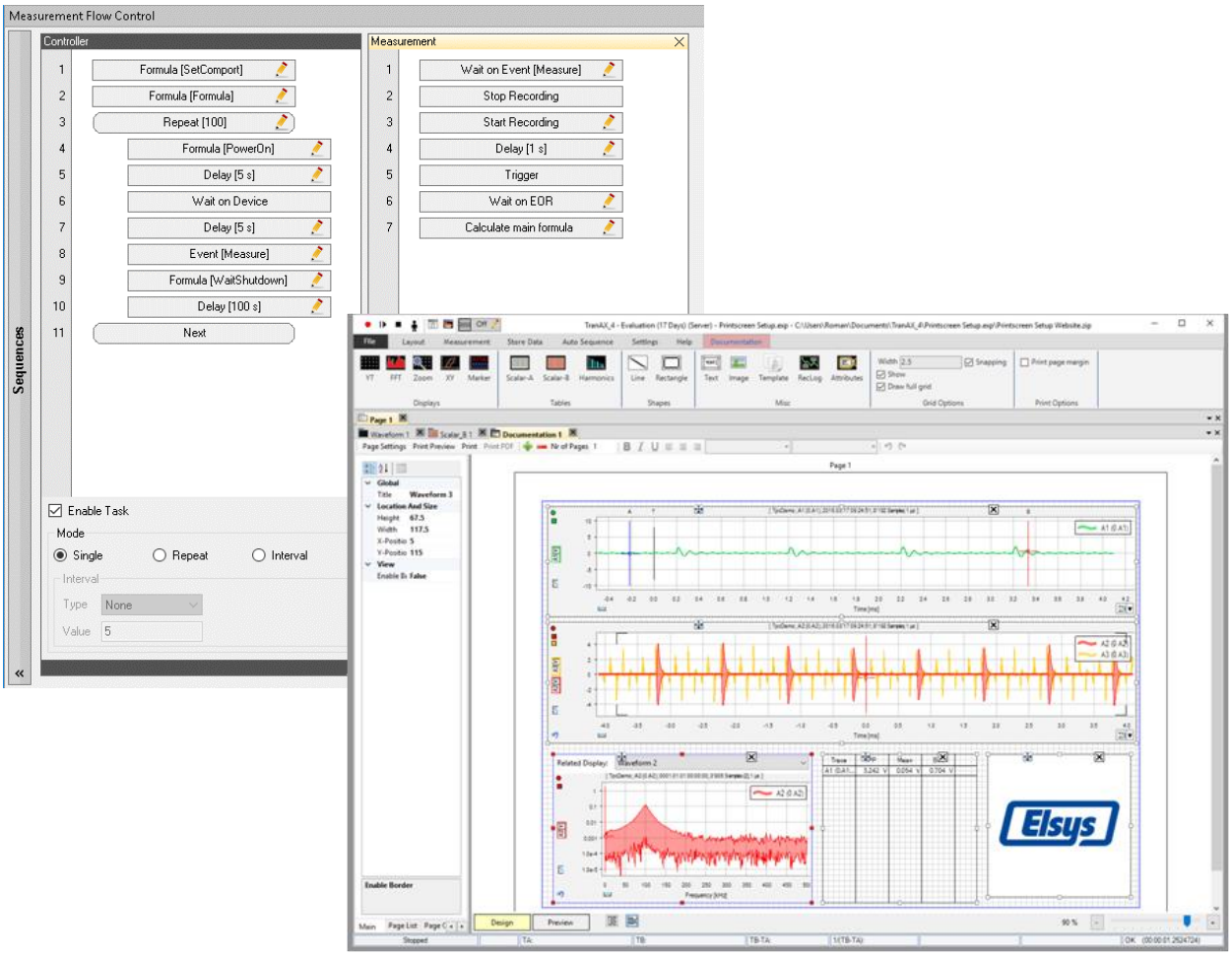
TranAX 4.1

Messdatenerfassung und Analyse



Benutzerhandbuch

Elsys AG
Mellingerstrasse 12
CH-5443 Niederrohrdorf

+41 56 496 01 55
www.elsys-instruments.com

Inhaltsverzeichnis

1 Einleitung und Neues

1.1 Vorwort

Betreiben Sie unsere **leistungsstarke Hardware**, TraNET, TPCX oder TPCE Module, mit der **Datenerfassungs- und Analyse-Software TranAX**. Dieses Benutzerhandbuch zeigt Ihnen alle TranAX Funktionen, Betriebsmodi und Einstellungen. Es gibt Ihnen einen Überblick über das System und dessen umfangreiche Fähigkeiten.

Schnell und einfach können viele Aufzeichnungskanäle konfiguriert und für die Messung vorbereitet werden. Alle Einstellungen können gespeichert und wieder geladen werden. Dies ermöglicht Layouts und Hardware Einstellungen im Vorfeld der Messung vorzubereiten und mit verschiedenen Anwendern zu teilen.

Funktionalität

- Einfache Programmierung der Kanäle **ohne Programmierkenntnisse**
- Datenvisualisierung in Multi-Waveform Displays
- Mehrere Cursors ermöglichen einfaches Daten ablesen und Dokumentation
- X-Y Darstellung
- FFT Analysis mit verschiedenen Skalierungen und Fensterfunktionen
- Messdaten - **Video Synchronisierung**
- **TranAX SCOPE vereinfachtes Kurvenfenster**
- Mehr als **40 Skalar Funktionen** auf Zeit- und FFT-Kurven
- Formeleditor mit mehr als **100 Mathematik Funktionen**, Syntax Highlighting, For-Schlaufen, If/else Bedingungen, Array Berechnungen, etc.
- Kurven fitten (Polynomiale Regression)
- **Makrosprache** für die Messablaufsteuerung
- Deutsche und englische Version
- Datenexport im **HDF5**, ASCII und Kundenspezifischen Datenformaten
- Datenimport von ASCII oder TPC (TransAS 2)
- Import und Export von Audiodateien (*.wav und *.mp3)
- **ActiveX/COM Schnittstelle** für das Steuern von TranAX und Auslesen der Daten von Excel oder kundenspezifischen Programmen

TranAX besteht aus einzelnen Fenstern welche via Drag & Drop verschoben und platziert werden können. So besteht die Möglichkeit die Oberfläche nach den Bedürfnissen des Anwenders zu gestallten.

In TranAX taucht oft das Wort "**Experiment**" auf. Mit diesem **Begriff** ist eigentlich ein **Projekt** gemeint. Darin werden Einstellungen wie Messbereich, Abtastrate, Kanalnamen, die Anordnung der Fenster, Formeln, Autosequenzen usw. für einen bestimmten Messvorgang gespeichert. Ebenso werden normalerweise gespeicherte Signale innerhalb dieser Experiment-Umgebung verwaltet.

Falls TranAX für Sie neu ist, empfehlen wir Ihnen beim Kapitel Erste Schritte zu beginnen.

1.2 Vollversion vs. Light Edition vs. Testversion

Es existiert nur eine Version von TranAX 4.1. Nach der Installation kann diese für 30 Tage als Vollversion verwendet. Nach dieser Zeit werden die Funktionen reduziert und die Software kann auf unbegrenzte Zeit als Light Edition (LE) weiterverwendet werden. Um die Vollversion zu erhalten, kann TranAX jederzeit mit einem Lizenzschlüssel aktiviert werden.
Die Testversion, welche auf der Seite www.elsys-instruments.com heruntergeladen werden kann, läuft wie oben beschreiben für 30 Tage als Vollversion und schaltet ohne Lizenzschlüssel anschliessend in Light Edition (LE).

Vorteile der Vollversion:

- Measurement Flow Control (MFC)
- Virtual Hardware Manager (VHM)
- Unlimitierte Anzahl Pages und Waveforms
- Alle verfügbaren Funktionen im Formel Editor

1.3 Systemanforderungen

TranAX lässt sich auf jedem üblichen Microsoft Windows basierten Computer installieren und verwenden. Die folgenden Kriterien sollten erfüllt werden, um das Maximum an Effizienz und Ergonomie zu erhalten;

- Microsoft Windows OS, Windows 7 oder neuer, 32bit oder 64bit Version
- Installation benötigt etwas 250MB, es sollten etwa 1GB freier Speicherplatz auf der Festplatte oder SSD vorhanden sein.
- 4GB RAM, bitte beachten, dass für mehr Speicher Windows 64bit verwendet werden muss.
- Bildschirmauflösung 1366x768, höhere Auflösung wird empfohlen.

1.4 Was ist neu an dieser Version?

TranAX 4.1 bietet viele neuen Funktionen, Innovationen und Erweiterungen zur Vorgänger Version.

1.4.1 Measurement Flow Control (MFC)

MFC ersetzt die bekannten und oft benutzten Autosequenzen der Vorgängerversionen. MFC bietet zusätzlich viele neue hilfreiche Funktionen. MFC ist das ideale Werkzeug um Messabläufe zu automatisieren. Mit den vordefinierten Massabläufen gehen keine Messungen aus Versehen mehr verloren.

Messablaufsteuerung (MFC) bieten verglichen mit den Autosequenzen folgenden Vorteile:

- Mehrere parallellaufende Tasks (Abläufe)
- Ausführung der Task, als Einzeldurchlauf (Single), Wiederholend (Repetition) oder auch als Intervall einstellbar
- Grafische Zusammenstellung der Task mit Drag & Drop Elementen
- Einfache Ablaufprogrammierung mit if/else Statements
- Einzelne Formeln innerhalb der Tasks ausführbar
- Tasks können per programmierbaren Events ausgelöst werden, wie z.B. "End Of Recording", "Trigger" oder vordefinierte Zeitspannen oder anhand der gemessenen Datenblöcke
- Grafische Darstellung des Messablaufs, aktuelle Position im Ablauf
- Schnelle Ausführung der einzelnen Tasks und Sequenzen
- Mit MFC kann auch die Automation mit Drittanbieter Hardware und Software realisiert werden

1.4.2 Virtual Hardware Manager (VHM)

Der Virtuelle Hardware Manager (VHM) stellt einen Software Layer dar, welcher zwischen die Anwendung wie TranAX oder BallAX und der eigentlichen Hardware, sprich TPCServer geschalter werden kann.
Mit dem VHM können so virtuelle Kombinationen von Elsys DAQ Messkarten und Geräten erstellt werden. Ist keine Hardware vorhanden, kann diese als Simulierte Komponenten vorberietet werden. So können auch ganze Experimente und Einstellung für spätere Messungen losgelöst von Hardware erstellt werden.

1.4.3 Zoomfunktionen

Zoomfunktionen in den Waveforms wurden optimiert, bekannten Kombinationen wie die Tasten "Ctrl" und Scrollrad rauf und runter, sowie Shift und Scrollrad rauf und runter für Links und Rechts werden neu unterstützt.
Die Tastenkürzel Shift X und Shift Y erlauben es das zoomen mit der Maus in nur eine Achse zu erlauben und die andere zu sperren.

1.4.4 Portable Version

Der TranAX Installer ist nun auch als Portable Version verfügbar. So kann TranAX auch ohne Administrationsrecht in einen entsprechenden Ordner, auf welchem Schreibrechte vorhanden sind, installiert werden. Dies ist nützlich zu Testzwecken und Evaluationen.

1.4.5 Wissensdatenbank WIKI

Im Elsys WIKI können weitere hilfreiche Informationen gefunden werden. Beispiele, detaillierte Anwendungen und weiteres Fachwissen zu unseren Produkten und Applikationen werden dort hinterlegt. Häufig verwendete Beispiele zum Formel Editor sind dort ebenfalls zu finden.
http://wiki.elsys-instruments.com/index.php

1.4.6 Weitere neue Funktionen

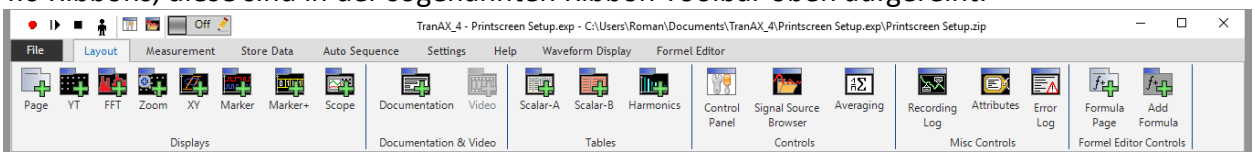
- Geschwindigkeitsoptimierungen wurden vorgenommen: Schnellere Scalar Tabellen sowie das parallele Abarbeiten von Prozessen und Formeln auf Systemen mit mehreren Kernen.
- Neue Formelfunktionen für schnelleren, effizienteren und einfacheren Code
- Neuen Funktionen für die vereinfachte Handhabung mit Datum- und Zeitvariablen

1.5 Was war neu an der Vorgängerversion?

TranAX 4.0 setzt neue Massstäbe in Punkto Benutzerfreundlichkeit, Bedienung und Funktionsumfang.

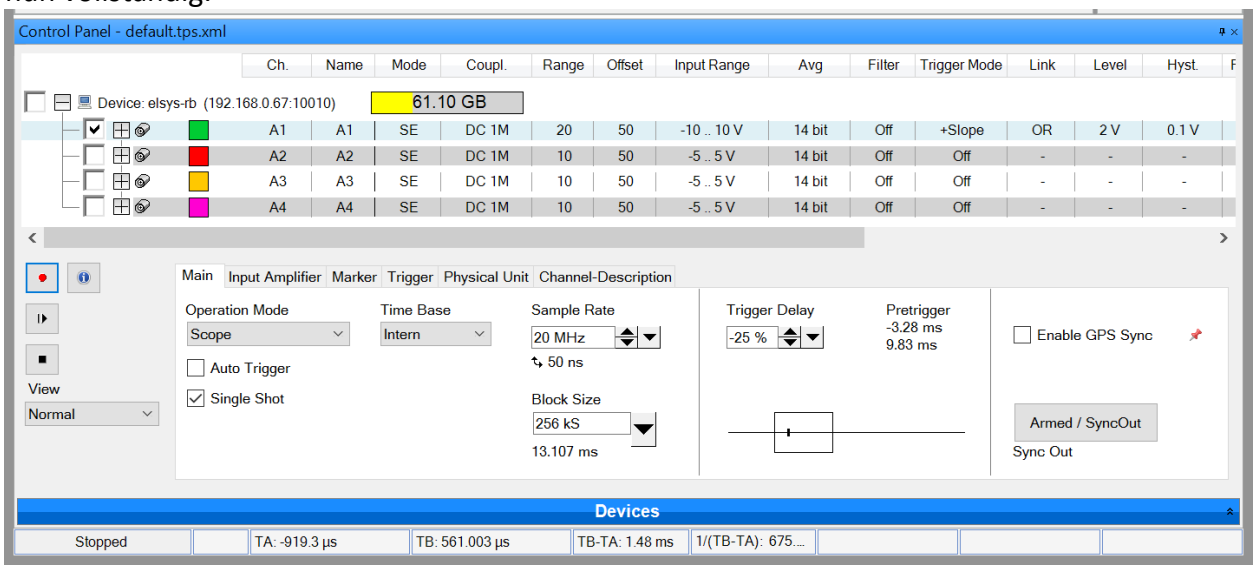
1.5.1 Ribbon Toolbar

Um klarere Strukturen in der Bedienung und Funktionalität zu erlangen, nutzt TranAX ab Version 4.0 Ribbons, diese sind in der sogenannten Ribbon Toolbar oben aufgereiht.



1.5.2 Support für Hochauslösende Displays

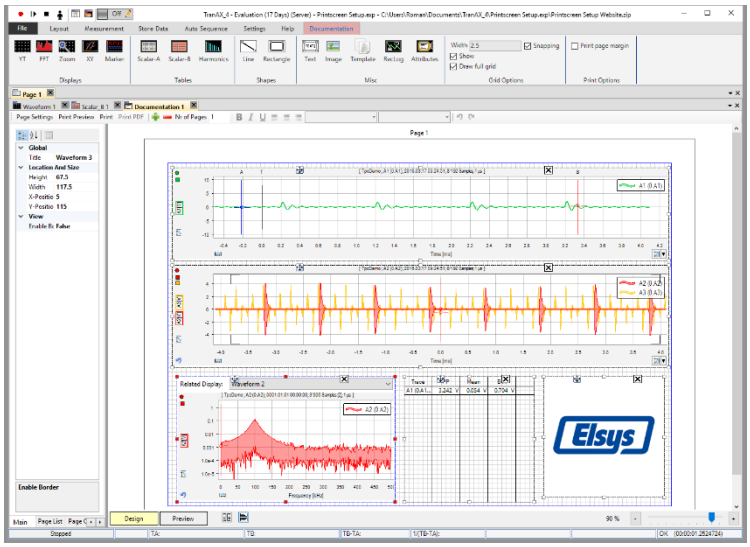
Bei neuen Systemen und Notebook ist die Skalierung der Fonts höher Skaliert als die Standard 96 DPI, meist auf 120% oder noch höher. Die neuen Komponenten in TranAX 4.0 unterstützen diese nun vollständig.



1.5.3 Dokumentation Waveform

Das Ausdrucken eines Messberichts ist für viele Kunden nach wie vor ein wichtiges Bedürfnis. In TranAX 3 war es ziemlich schwierig, die Messdaten in einem vordefinierten Format auszudrucken, da dies von der Bildschirmgröße und der Auflösung abhing.
Daher wurde in TranAX 4 ein WYSIWYG-Dokumentationsfenster (Was Sie sehen, ist was Sie erhalten) implementiert. Es ermöglicht das freie Platzieren von Wellenformkurven, Skalartabellen, Bildern und Textelementen wie auf einem Blatt Papier. Kurz gesagt, einige der neuen Möglichkeiten:

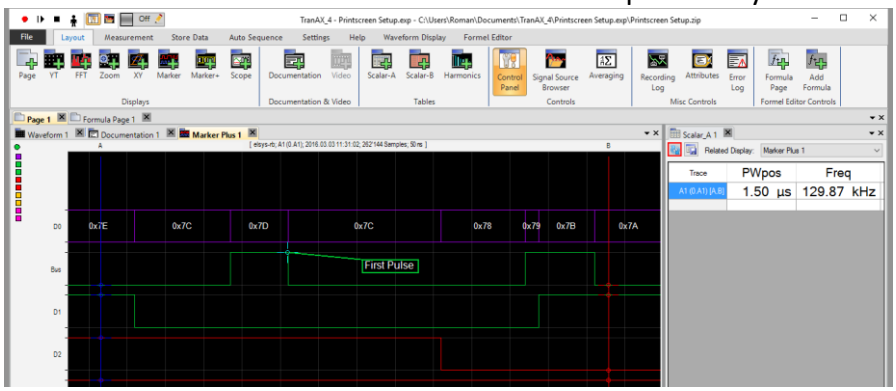
- Erstellen Sie mit nur wenigen Klicks einen Messbericht.
- Teilen Sie Langzeitmessungen bei Bedarf auf mehrere Seiten auf.
- Verwenden Sie Berichtsvorlagen, um ein einheitliches Erscheinungsbild und die CI-Konformität Ihrer Berichte zu erzielen.
- Es sind keine unterschiedlichen Layout-Einstellungen mehr für die Visualisierung der Messdaten auf dem Bildschirm und die Berichterstellung zum Drucken erforderlich.
- Berichtsergebnisse werden vom Formeleditor direkt in einem Textfeld ausgegeben.
- Verwenden Sie die MFC-Funktion "Print", um den Bericht nach jeder Messung auszudrucken oder in einer PDF-Datei auszudrucken.



1.5.4 Digital Signals (Marker) Waveform

Für die Analyse digitaler Signale - in TranAX Marker genannt - steht ein neues Marker-Fenster zur Verfügung. In diesem Fenster werden Marker-Signale wie in der normalen Waveform für die analogen Eingänge als einzelne digitale Signale behandelt. Einige Funktionen sind:

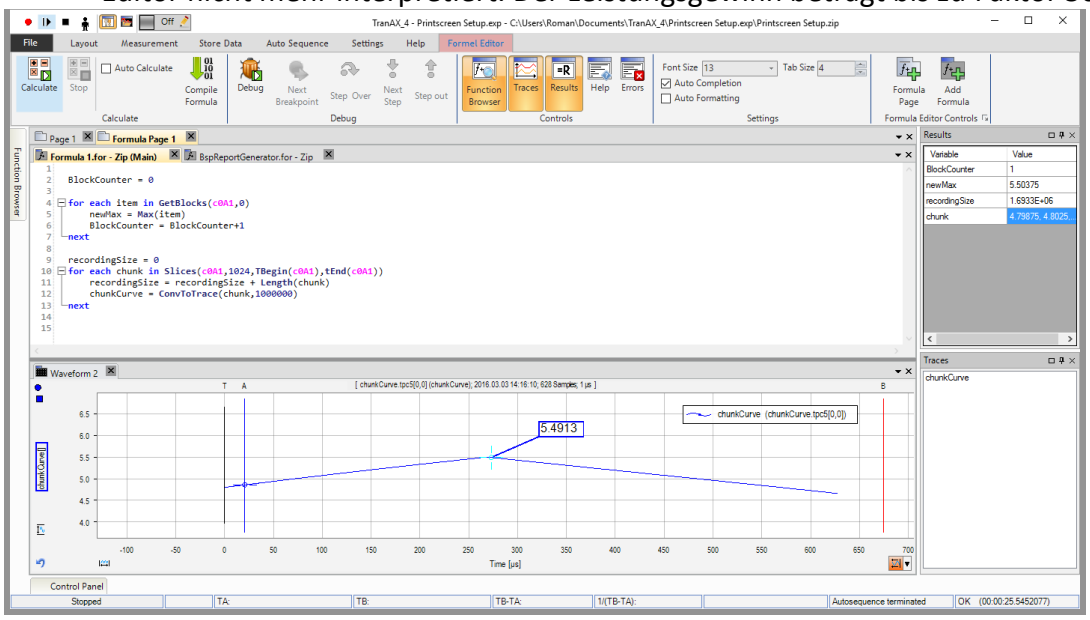
- Die Anordnung der Marker kann unabhängig vom entsprechenden zugehörigen Analogkanal geändert werden.
- Verschiedene Markierungen können als Bus gruppiert werden.
- Numerische Interpretation von Bussen mit unterschiedlichen Anzeigeformaten und Bitreihenfolge
- Skalar Funktion zur Pulsweiten- und Frequenzanalyse



1.5.5 Formel Editor

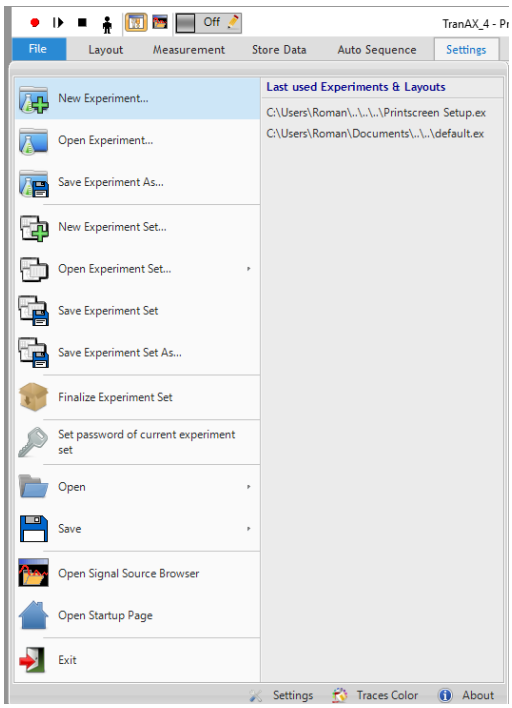
Dem Formeleditor wurden mehrere neue Funktionen hinzugefügt:

- Der Formeleditor wird jetzt in einer Registerkarte wie einem Waveform Window angezeigt, genau wie in der Ergebnistabelle. Dies ermöglicht es, die Ergebnisse flexibler und näher an anderen Teilen des Programms zu platzieren.
- Mit dem neuen Element "für jedes" können Blöcke oder Slices wesentlich komfortabler durchlaufen werden. Die Analyse großer Messungen ist jetzt viel einfacher.
- Vordefinierte Skalar Berechnungen wie Pulsbreite oder Peak-Peak aus der Skalartabelle sind jetzt auch im Formeleditor als Funktion verfügbar.
- Komplexe und zeitaufwändige Berechnungen können durch Kompilieren der Formel beschleunigt werden. Dieser Schritt konvertiert die Formel in eine DLL und wird daher vom Editor nicht mehr interpretiert. Der Leistungsgewinn beträgt bis zu Faktor 50.



1.5.6 Layout und Hardware Settings

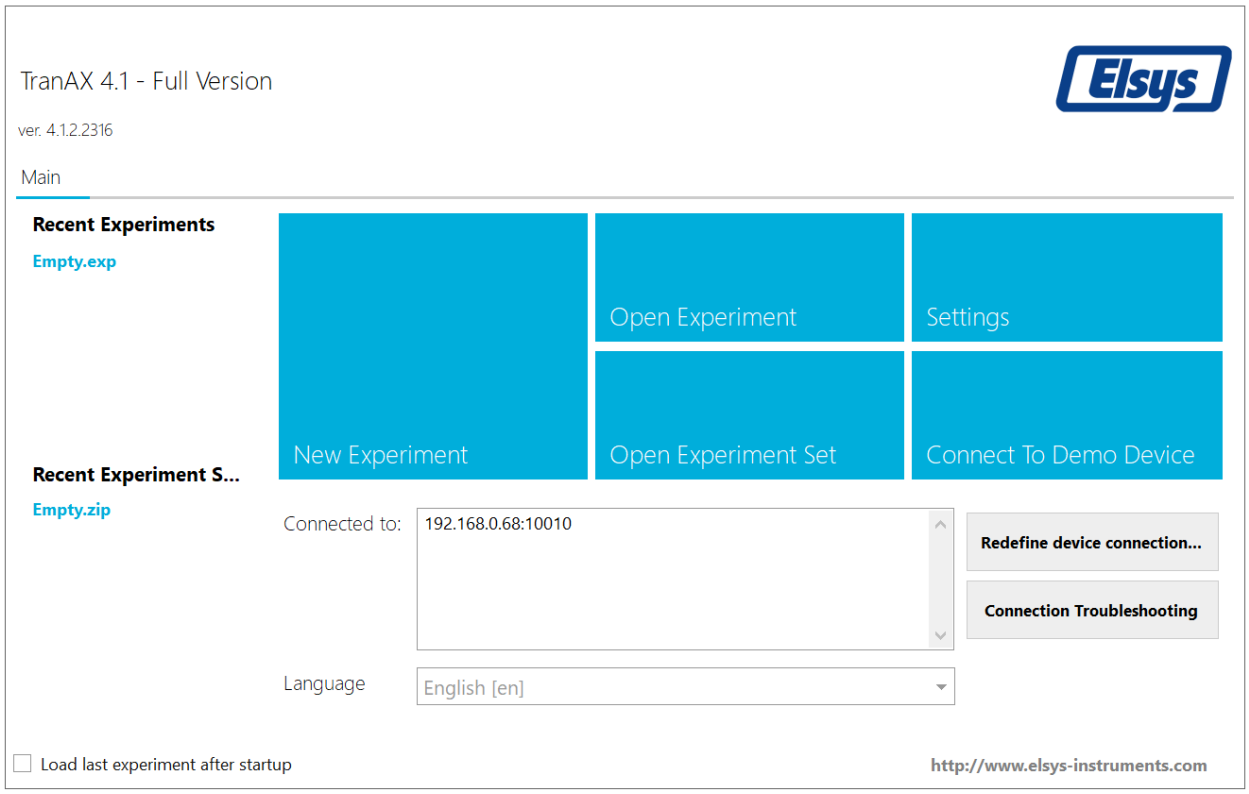
Experimente und Einstelllungen von TranAX 3 und TranAX 4 sind zu 100% kompatibel. Alle bestehenden Experimente können weiterhin verwendet werden, neue Experimente lassen sich auch mit der alten Software verwenden.



Trotzdem wurden einige Verbesserungen vorgenommen, welche das Handling und die Produktivität steigern werden. Experimente sind neu in sogenannten Experiment Sets unterteilt, welches als ZIP-Archiv abgelegt wird. Dieses beinhalten wie gehabt die einzelnen Einstellungen von Layout, Hardware Settings und Formeln. Diese Struktur bietet den Vorteil, dass beim Kopieren und Archivieren immer alle Dateien zusammenbleiben.

1.5.7 Start Screen

Der Start Screen zeigt eine Liste mit den zuletzt verwendeten Experimenten und Einstellungen. Dieser bietet auch gleich die Möglichkeit neue Experimente aus Vorlagen zu erstellen.
Im Menu *"Datei / Startseite öffnen"* kann dieser Dialog jederzeit wieder angezeigt werden.



2 Aufbau und Komponenten

TranAX lässt sich die die folgenden Hauptkomponenten unterteilen:

2.1 Menu Bar

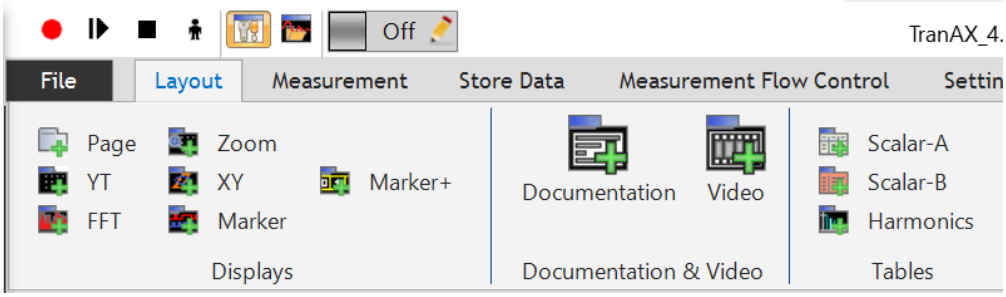
Hier ist der Schnellzugriff auf die wichtigsten Bedienelemente möglich: Messung- starten, stoppen und manueller Trigger auslösen. Wichtige Elemente wie das Control-Panel und der Signalquellen-Browser können direkt geöffnet werden.

	Aufzeichnung starten (F6)
	Manueller Trigger (F7)
	Aufzeichnung manuell beenden (F8)
	Status-Anzeige der Datenerfassung Animation (Ergänzend zur gelben Anzeige links unten)
	Öffnet das Control-Panel
	Öffnet den Signalquellen-Browser. Dies ermöglicht den Zugriff auf Signale, auch aus Dateien
Off	Experiment Sets lassen sich schreibgeschützt speichern, so dass diese während der praktischen Anwendung nicht mehr geändert werden können.

Der Switch für Schreibschutz bietet die Möglichkeit, Einstellungen zu sichern, so dass diese beim nächsten Start von TranAX unverändert wieder geöffnet werden. Dies ist besonders für Serienprüfungen und Produktion von Vorteil um ungewollte Veränderungen der Einstellungen zu vermeiden.

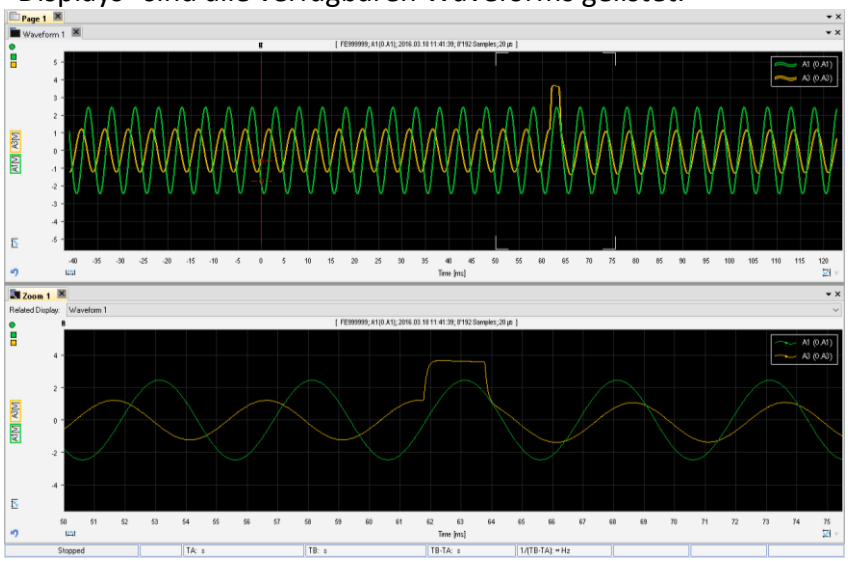
2.2 Ribbon Bar (Ribbon Leiste)

In der Ribbon Bar sind alle verfügbaren Funktionen und Einstellungen gelistet. Je nach selektiertem Waveform oder Page sind die entsprechenden verfügbaren Ribbon Tabs aktiv. Die Bedienung lehnt sich so stark an die von Microsoft Office Produkten und ähnlichen Anwendungen an.



2.3 Pages und Waveforms (Seiten und Kurvenanzeige)

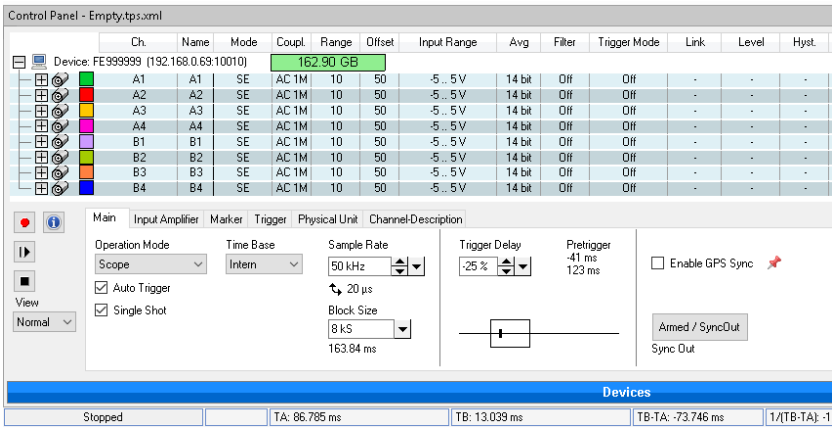
Auf den sogenannten Pages können die einzelnen Waveforms (YT, FFT, Zoom, etc.) platziert werden. Hier werden die Messdaten und Kurven dargestellt. Beim Ribbon Tab "Layout" in der Gruppe "Displays" sind alle verfügbaren Waveforms gelistet.



2.4 Control Panel

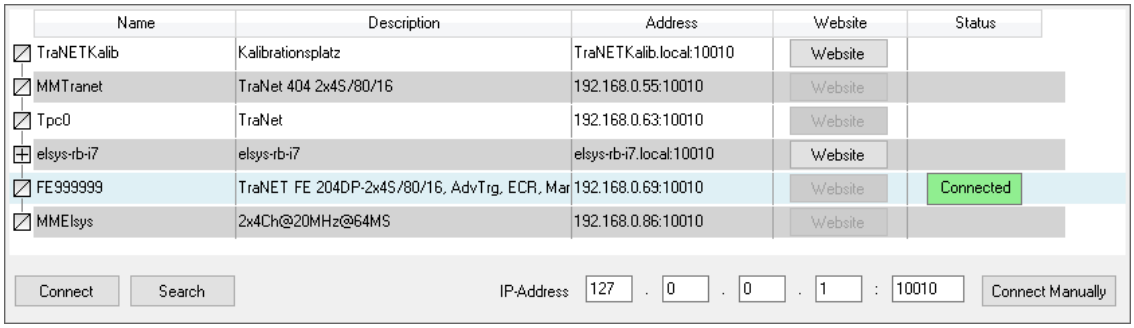
Das Control-Panel wird verwendet um die Einstellungen der Hardware (Messeinstellungen) vorzunehmen. Auch lassen sich hier die einzelnen Messmodule in Gruppen kombinieren, sogenanntes Clustern.

Es gilt zu beachten, dass geänderte Control-Panel-Einstellungen in der Regel erst auf die nächste Messung sich auswirken.



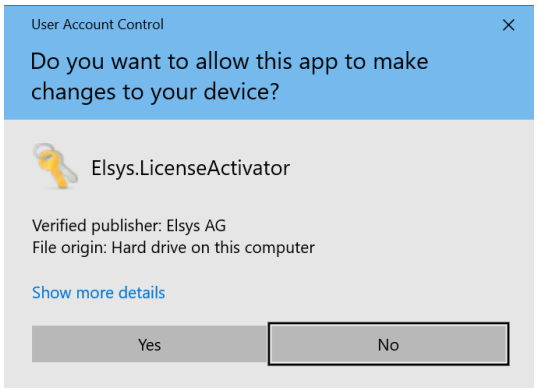
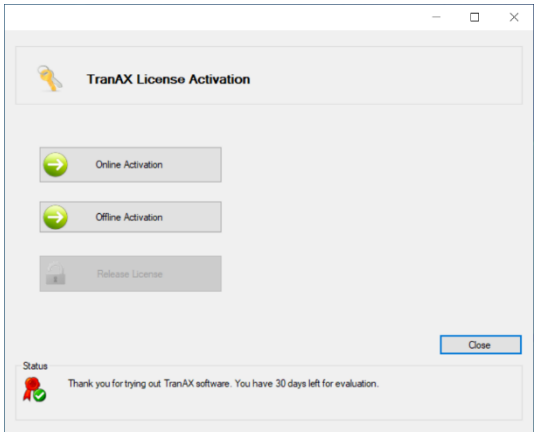
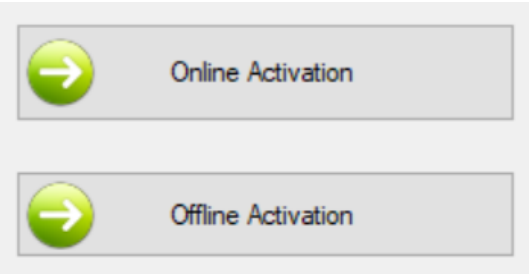
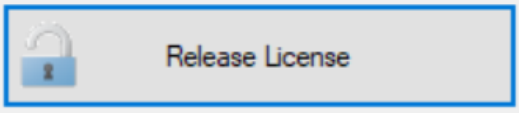
2.5 Devices Manager

Mit dem Devices Manager lassen sich sowohl einzelne als auch ganze Gruppen von TraNET Geräten einbinden. Diese können anschliessen im Control Panel weiter konfiguriert und bedient werden.



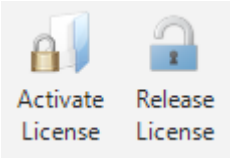
3 Lizenzverwaltung

Seit TranAX 4.0 ist eine Softwareaktivierung erforderlich, um die Vollversion von TranAX verwenden zu können. Ohne schaltet die Software nach einer Testphase von 30 Tagen in den LE-Modus (Light Edition).

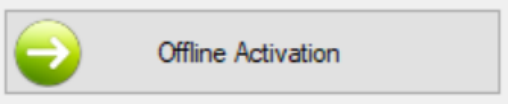
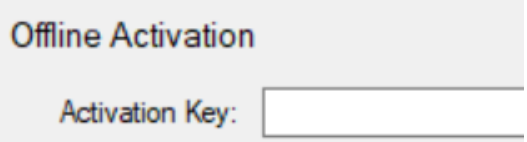
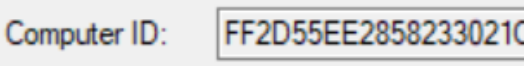
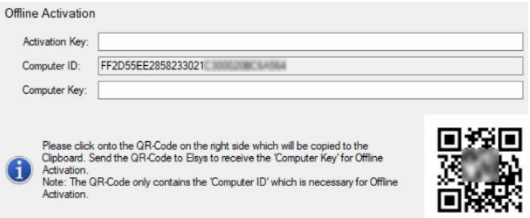
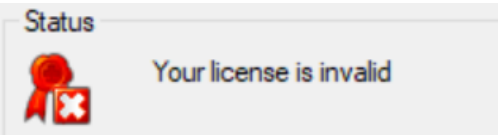
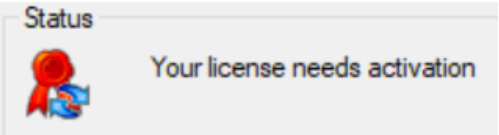
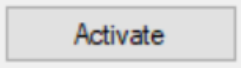
	Nach dem ersten Start erscheint der Dialog "TranAX Lizenzaktivierung". Wird die Aktivierung zu einem späteren Zeitpunkt ausgeführt, muss für diesen Administrationsrechte gewährt werden.
	TranAX funktioniert nun als Vollversion mit allen Features und Optionen wie "Com/ActiveX", "All Import Options" und "Excel-Report" für 30 Tage. Nach dieser Zeit muss TranAX aktiviert werden, ansonsten läuft TranAX im LE Modus (Light Edition) mit reduziertem Funktionsumfang weiter.
	Um alle implementierten Funktionen und Vorteile nutzen zu können, muss die Software aktiviert sein. Dies kann online und, falls keine Internetverbindung oder Einschränkungen bestehen, auch offline erfolgen.
	Soll die Lizenz auf einen anderen Rechner portiert werden (Neuer Rechner, Wechsel innerhalb einer Abteilung etc.), so kann die Lizenz wieder freigegen werden und auf dem neuen Rechner aktiviert werden.

TranAX benötigt nur für die erste Online Aktivierung eine Internetverbindung. Für die weitere Verwendung ist diese nicht mehr erforderlich!

Im Ribbon Tab "Einstellungen" kann die TranAX Lizenz jederzeit aktiviert oder auch wieder deaktiviert werden.

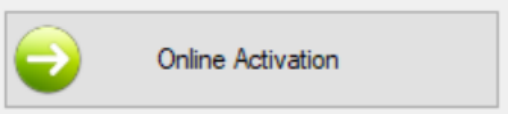
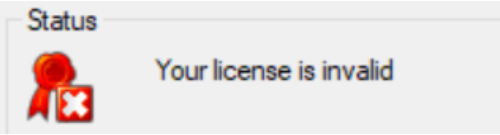
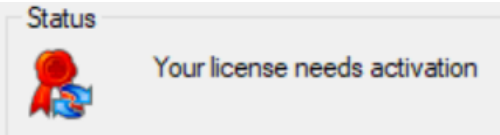
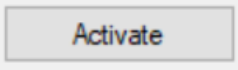


3.1 Offline Aktivierung

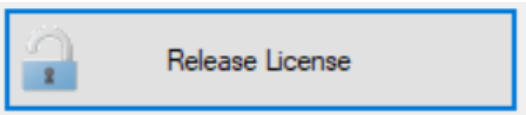
	Klicken Sie auf die Schaltfläche "Offline-Activation".
	Auch hier muss Im Feld "Lizenz Key" wird nun der Lizenz Code aus dem Handbuch oder dem USB Stick eingegeben werden.
	Zusätzlich muss die "Maschinen ID" zwecks Aktivierung an Elsys Instruments gesendet werden, per Email an info@elsys.ch.
	Es ist auch möglich, einfach auf den QR-Code zu klicken. Dadurch werden die Grafiken in die Windows-Zwischenablage kopiert und kann in eine E-Mail eingefügt.
	Ist die Lizenzanfrage berechtig, wird der passende "Maschinen Key" zugestellt kann in das entsprechende Feld eingetragen werden.
	Im Falle eines Typfehlers oder eines anderen Grundes für einen ungültigen Aktivierungsschlüssel ist eine Information im Status sichtbar.
	Wenn der Aktivierungsschlüssel korrekt eingegeben wurde, sieht der Status folgendermassen aus.
	Klicken Sie auf die Schaltfläche "Aktivieren" und TranAX wird als Vollversion aktiviert. Bitte beachten Sie, dass keine Internetverbindung mehr erforderlich ist.

Der QR-Code enthält nur die "Computer ID", welche für die Offline-Aktivierung erforderlich ist.

3.2 Online Aktivierung

	Klicken Sie den Knopf "Online Activation".
	Im Feld "Activation Key" wird nun der Lizenz Code aus dem Handbuch oder dem USB Stick eingegeben und durch drücken des Knopfs "Aktivieren" wird TranAX freigeschalten.
	Im Falle eines Typfehlers oder eines anderen Grundes für einen ungültigen Aktivierungsschlüssel ist eine Information im Status sichtbar.
	Wenn der Aktivierungsschlüssel korrekt eingegeben wurde, sieht der Status folgendermassen aus.
	Klicken Sie auf die Schaltfläche "Aktivieren" und TranAX wird als Vollversion aktiviert. Bitte beachten Sie, dass keine Internetverbindung mehr erforderlich ist.

3.3 Lizenz deaktivieren

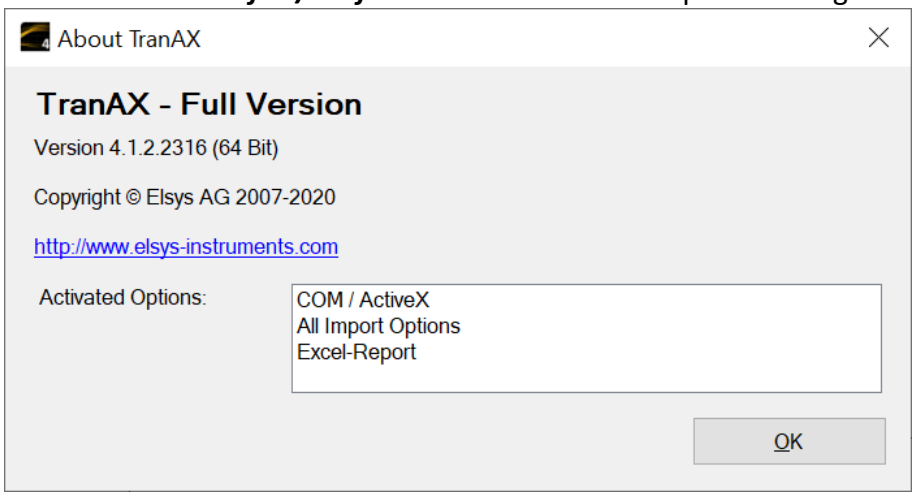
 	Soll die Lizenz auf einen anderen Rechner portiert werden (Neuer Rechner, Wechsel innerhalb einer Abteilung etc.), so kann die Lizenz wieder freigegeben werden und auf dem neuen Rechner aktiviert werden.

3.4 Software Optionen Installieren

Bevor eine Option verwendet werden kann, muss diese **aktiviert** werden. Die Optionen sind mit dem Lizenzschlüssel gekoppelt und werden durch den Elsys Lizenzserver verwaltet. In TranAX sind zurzeit folgende Optionen verfügbar:

Name	Beschreibung
Import	Nachdem eine Import-Option installiert wurde, lässt sich im Ribbon Tab "Daten speichern" / "Importieren" eine der nachfolgenden Datei-Typen importieren. • *.TPC (aus TransAS 2) • *.ASD (ASCII aus TransAS 2 • *.MP3, *.WAV (Audio-Dateien) • *.SGY (Segy, geologisches Dateiformat) Des Weiteren können mit Hilfe eines sog. "Wizard" alle Arten von ASCII (Text) Dateien importiert werden. Beim Importieren werden die Fremddateien in das *.TPC5-Format kopiert und können danach in TranAX als solche verwendet werden.
ActiveX	Aktiviert die ActiveX Schnittstelle, dadurch kann TranAX von einem anderen Programm gesteuert werden.
Report-Generator	Diese Option erlaubt die Erzeugung von EXCEL-Berichten. Nachdem sie installiert wurde, kann im Formel-Editor auf die entsprechenden Funktionen zugegriffen werden.

Im Ribbon Tab ***"Hilfe" / "Info"*** sind die aktivierten Optionen aufgelistet.



Falls Sie eine Option nachträglich erwerben möchten, wenden Sie sich bitte an die für Sie zuständige Vertretung.

4 Ribbon Bar Elemente

Die Ribbon Bar besteht aus einzelnen Elementen, Tabs und Groups. Diese werden auf den folgenden Seiten beschrieben

4.1 Ribbon Tab "Layout"

Generell sind hier alle Elemente zu finden, welche sich in TranAX platzieren lassen.

4.1.1 Kurvenanzeige

Page	Erstellt eine neue Seite mit einer YT Waveform
YT	Fügt eine neue Kurvenanzeige hinzu
FFT	Fügt eine neue FFT-Kurvenanzeige hinzu. Diese zeigt das Frequenz-Spektrum des Signals an.
Zoom	Fügt ein neues Ein Zoom Waveform ein. Dieses bezieht sich immer auf ein bestehendes YT Waveform. Mit diesem kann einen gezoomten Teilbereich des Referenz Waveform dargestellt werden.
XY	Fügt ein neues XY Waveform ein. In diesem können mehre Messkurven der Y-Achse auf eine Bezugskurve in der X-Achse dargestellt werden.
Marker	Fügt ein neues Marker Waveform ein. Bei diesem können die letzten 2 Bit der Messkurve, welche bei 14Messungen als Marker verwendet werden, dargestellt werden.
Marker+	Fügt ein erweitertes Marker Waveform. Diese bietet erweiterte Möglichkeiten zur Analyse und Gruppierung einzelner digitaler Signale. Gruppierte Marker können auch als Bus-Signale dargestellt werden.
Scope	Öffnet ein neues SCOPE-Fenster

4.1.2 Documentation & Video

Documentation	Fügt ein eines Dokumentationsfenster ein. In diesem lassen sich Berichte und Rapporte erstellen, so wie diese dann auch in gedruckter Form aussehen werden. Aufnahmen werden zur Echtzeit in diesem Fenster aktualisiert.
Video	Fügt ein Video Fenster ein. Dies bietet die Möglichkeit, bestehende Video-Aufnahmen synchron mit der Messkurve abspielen zu können.

4.1.3 Tabellen

Scalar-A	Fügt eine Skalar Funktions-Tabelle A hinzu. Skalar Tabellen A sind dazu ausgelegt, die gleichen Messwerte aus mehreren Kurven zu lesen.
Scalar-B	Fügt eine Skalar Funktions-Tabelle B hinzu. Skalar Tabellen B können dazu verwendet werden, einzelne Messwerte aus bestimmten Kurven zu lesen.
Harmonics	Fügt eine Tabelle der Harmonischen hinzu. Die Tabelle der Harmonischen kann in Kombination mit einem FFT Waveform zur Frequenzanalyse verwendet werden.

4.1.4 Controls

Control Panel	Öffnet das Control Panel, in diesem werden die Einstellungen zur Aufnahme der Messungen und die Einstellungen der einzelnen Kanäle vorgenommen.
Signal Source Browser	Öffnet den Signal Source Browser. Hier können bestehende Messkurven zur weiteren Analyse geladen werden. Auch Referenzen, einzelne Blöcke der Aufnahmen werden hier ausgewählt.
Averaging	Öffnet das "Averaging" Fenster. Mit diesem lassen sich die Berechnung und Mittelung mehrere Aufnahmen zur reduzieren von Störungen einstellen und durchführen. Diese Funktion kann nur in Kombination mit einer Scope-Aufnahme verwendet werden.

4.1.5 Diverse Controls

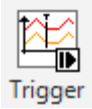
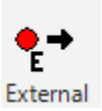
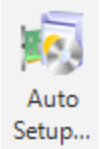
Recording Log	Öffnet das Aufnahmeprotokoll Fenster. Recording Logs können jeweils in der Kurvenanzeige als Textmarkierungen platziert werden. Jedes Recording Log hat einen genauen Zeitstempel. Die einzelnen Recording Logs erscheinen dann im Aufnahmeprotokoll Fenster.
Attributes	Öffnet das Attribut Fenster. Attribute können verwendet werden um Projektbezeichnungen, Versuchsteilnehmer oder Hinweise zum Messaufbau zu protokollieren.
Error Log	Öffnet das Fehlerprotokoll Fenster. Hier werden Softwarefehler protokolliert.

4.1.6 Formel Editor Controls

Formula Page	Öffnet Eine neue Formeleditor Seite.
Add Formula	Fügt der Seite einen neuen Formel Editor hinzu.

4.2 Ribbon Tab "Messung"

4.2.1 Messung

	Startet eine neue Messung (F6).
	Löst einen manuellen Trigger im Messsystem aus (F8).
	Stoppt die aktuelle Messung.
	Aktiviert oder deaktiviert das Auslösen der Messung eines externen Signals.
	Öffnet das Auto Setup Fenster. Hier kann automatisch der Messbereich ermittelt werden anhand der anliegenden Signale.
	Aktiviert oder deaktiviert die Benachrichtigung von nicht gespeicherten Messungen falls das Programm beendet oder eine neue Messung gestartet wird.

4.3 Ribbon Tab "Daten speichern"

4.3.1 Daten speichern und laden

Save Tpc5	Speichert Signalkurven (Zeitbereich) im TPC5 Format. Das Datei-Format entspricht dem HDF5-Standard.
Save Tps5	Speichert Spektrumkurven (Frequenzbereich) im TPS5 Format. Die Spektrum-Kurven müssen vorher in einem FFT-Fenster erzeugt worden sein.
Save Page	Speichert eine komplette Seite. Dabei können die Quellen-Referenzen der angezeigten Kurven substituiert werden, sodass sie danach auf die entsprechenden Kurven in der Datei zeigen.
Load Page	Lädt eine komplette Seite.

4.3.2 Importieren

Import Traces	Öffnet den Importdialog. Hier können diverse Formate importiert werden (*.asd, *.tpc (TransAS2), *.mp3, *.wav, *.sgy).
Ascii Wizard	Öffnet die ASCII Wizard. Dieser Dialog ermöglicht ASCII-Formate von Drittprogrammen zu importieren.

4.3.3 Exportieren

Export Traces	Öffnet den Exportdialog. Hier können diverse Formate exportiert werden (ASCII, Krenz, Segy, DIAdem, TransAS2, Wave). Weitere Formate können auf Anfrage implementiert werden.
Export Scalar Table	Exportiert die Tabelle in eine Textdatei (*.txt). Kolonnen sind jeweils mit einem Tabulator getrennt.

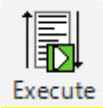
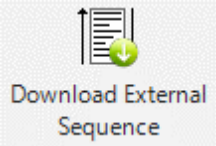
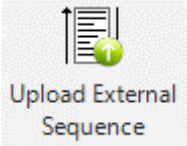
4.3.4 Drucken & Snapshot

Print Preview	Öffnet die Druckansicht für eine einzelne Kurvenanzeige. **Achtung! Diese Funktion ist veraltet, bitte verwenden sie das Dokumentationsfenster. Dieses bietet mehr Möglichkeiten in Bezug auf Darstellung und Druckbarkeit!**
Print	Öffnet den Druckdialog für eine einzelne Kurvenanzeige.

	Achtung! Diese Funktion ist veraltet, bitte verwenden sie das Dokumentationsfenster. Dieses bietet mehr Möglichkeiten in Bezug auf Darstellung und Druckbarkeit!
	Erstellt eine Bildschirmaufnahme eines einzelnen Elements (Kurvenanzeige, Skalar Tabelle, etc.) Im Ribbon Tab "Einstellungen", Schaltfläche "Einstellungen" bei "Benutzeroberfläche / Snapshot" können diverse Einstellungen dazu vorgenommen werden.

4.4 Ribbon Tab "Messablaufsteuerung"

4.4.1 Messablaufsteuerung

	Startet die definierten und aktivierten Tasks. Diese vorab erstellt werden, passend zu der gewünschten Anwendung und Funktionalität. Die **Taste F4** startet diese ebenfalls, wie zuvor die Autosequenz.
	Stoppt MFC und die definierten und aktiven Tasks. Die **Taste F8** stoppt diese ebenfalls, wie zuvor die Autosequenz.
	Öffnet das die Messablaufsteuerungsfenster (MFC).
☑ Autostart on Experiment Set loading	Wenn angewählt, starten die MFC Tasks beim Start von TranAX oder beim Laden eines Experiment Sets automatisch.
☑ Show current calculating sequence	Wenn angewählt, sieht man in den einzelnen Tasks farblich hervorgehet, bei welchem Schritt dieser im Moment steht. Dies benötigt zusätzlich Ressourcen und sollte nur zu Testzwecken und Entwicklung verwendet werden. Deaktiviert bringen die MFC Tasks die maximale Performance.
☑ Remote Measurement Flow Control	Wenn angewählt, wird der Tasks 1 direkt auf einem TraNET Gerät und nicht wie üblich in TranAX ausgeführt. Dies kann zur autonomen Messung und Überwachung von Systemen verwendet werden, Messdaten können so direkt auf die interne SSD gespeichert werden. Es werden nicht alle MFC Funktionen als Autonome Tasks unterstützt.
	Lädt einen MFC Task von einem TraNET FE Gerät in TranAX TraNET FE -> TranAX
	Lädt einen MFC Task von TranAX zu einem TraNET Gerät. TranAX -> TraNET FE

4.4.2 Editieren

Add Task	Erstellt einen neuen Task in im MFC. Jeder Task kann losgelöst von den anderen, aber auch mit diesen verknüpft verwendet werden. Alle Formel Berechnungen sind für jeden Tasks gekapselt. Um Daten von einem Task zu einem zu transferieren oder Verfügbar zu machen, müssen die Formel Funktionen **WriteforNext** und **ReadofPrevious** verwendet werden.
Remove Task	Löscht den ausgewählten Task. Der erste Task, der Haupttask kann nicht gelöscht, jedoch umbenennt werden.
Remove Sequence	Löscht die angewählten Kommandos eines Tasks. Alternativ kann auch die Taste " Delete " gedrückt werden.
Clear Sequences	Löscht den Inhalt eines Tasks. Dasselbe kann erreicht wenden, in dem alle Kommandos des Tasks ausgewählt werden und die Taste "Delete" gerückt wird.

4.4.3 Ansicht

Single Row	Alle Tasks werden in einer Reihe nebeneinander dargestellt, von links nach rechts.
Two Rows	Stellt alle Tasks in zwei Reihen dar. Nummerierung ist von oben nach unten und von links nach rechts.

4.5 Ribbon Tab "Einstellungen"

4.5.1 Einstellungen

Settings	Öffnet den Einstellungsdialog. Hier können diverse TranAX Einstellungen vorgenommen werden (Import/Export, Benutzeroberfläche, Systemleistung, Andere Einstellungen)
Trace Colors	Öffnet den Kanalfarbendialog. Hier können jedem Hardwarekanal eine Farbe zugewiesen werden.
Custom Toolbar	Öffnet einen Dialog für die benutzerspezifische Toolbar. Hier kann ein eigener Ribbon Tab erstellt werden, welcher diverse Buttons mit Ereignissen beinhaltet. Ein Ereignis kann beispielsweise ein spezifisches Layout laden.

Activate License	Öffnet den Lizenzdialog. Hier kann man eine Lizenz online oder offline aktivieren.
Release License	Falls eine Lizenz auf einem anderen Computer benötigt wird, kann man diese hier deaktivieren. Falls sie nicht online sind (bzw. keinen Internetanschluss haben), kann die Lizenz auf dem Lizenzserver nicht automatisch deaktiviert werden. In diesem Falle müssen sie Ihren Software Vertreiber kontaktieren.
Get Update	Startet den Download einer neuen TranAX Version.
Bug Report	Öffnet den Bug Report Dialog. Hier können Fehler gemeldet werden, die nicht provoziert werden können oder die ein Fehlverhalten von TranAX beschreiben.

4.6 Ribbon Tab "Hilfe"

4.6.1 Hilfe

Content	Öffnet das Benutzerhandbuch als PDF-Dokument oder zeigt eine Auswahl an mehreren vorhandenen Handbüchern.
Shortcuts	Öffnet einen Dialog, welcher alle Shortcuts für TranAX darstellt.
About	Zeigt Information über die Installierte TranAX Version und Optionen in einem Dialog

4.7 Ribbon Tab "Kurvenanzeige"

Wenn ein Waveform aktiviert ist, erscheint das zusätzliche Ribbon Tab "Waveform Displays"

Waveform Display

4.7.1 Block & Cursor Funktionen

Previous	Springt zum vorherigen Block.
Next	Springt zum nächsten Block.
Lock on screen	Verknüpft das Cursor-Paar in der Kurvenanzeige. Auch beim Zoomen und Verschieben der Kurven bleiben diese an ihrer Position der Kurvenanzeige.

Lock Timewindow	Arretiere Zeitfenster auf dem Bildschirm. Die Zeitfenster-Marken im Hauptfenster bleiben stehen, auch beim Zoomen und Verschieben der Kurven (sie sind nicht an die Kurven gekoppelt)
Fit Timewindow	Setzt die Zeitfenstermarken auf die Grenzen eines Blocks.
Fit Block to screen	Setzt die Zeitachse so, dass die Startzeit und Endzeit eines Blockes gerade der Größe der Kurvenanzeige entsprechen.
Move cursor simultaneously	Bewegt die Cursors gleichzeitig. Dadurch werden die Cursors (in der Regel A und B) gemeinsam verschoben

4.7.2 Sichtbare Elemente

☑ Cursors ☐ Horizontal Cursors ☑ Legend	Cursors sichtbar/unsichtbar Horizontale Cursor sichtbar/unsichtbar Legende sichtbar/unsichtbar
☑ Grid ☐ Block Numbers ☑ Trigger Lines	Raster in Kurvenanzeige sichtbar/unsichtbar Blocknummerierung sichtbar/unsichtbar Trigger Linien sichtbar/unsichtbar

4.7.3 Bereiche / Achsen / Cursors

Areas 1 Rulers Left 1 Rulers Right 0	Anzahl Bereiche Anzahl Y-Achsen auf der linken Seite Anzahl Y-Achsen auf der rechten Seite
Cursors 0	Anzahl zusätzliche Cursors (C-Z)

4.8 Ribbon Tab "Scalar Table A"

Wenn eine Skalar Tabelle A aktiviert ist, erscheint das zusätzliche Ribbon Tab "Scalar Table A"

Scalar Table A

4.8.1 Skalar Einstellungen

Function Cursor Amplitude	Auswahl der Skalar Funktion
Waveform A2 (0.A2)	Auswahl einer Referenzkurve. Beispielsweise für die Delta-Skalar Funktion.

4.8.2 Zahlenformatierung

Formatting None	Auswahl der Zahlenformatierung (None, Engineering, Fixed-Point, etc.)
No. Digits 3	Anzahl Stellen

4.9 Ribbon Tab "Formel Editor"

Wenn eine Formel Seite aktiviert ist, erscheint das zusätzliche Ribbon Tab "Formel Editor"

Formula Editor

4.9.1 Berechnung

Calculate	Mit der Schaltfläche Berechnen werden die Berechnungen manuell gestartet. Die Berechnung kann auch durch die Anweisung "Calculate" aus einer Autosequenz aufgerufen werden. Die Berechnung auch mit der Funktionstaste F10 eingeleitet werden.
Stop	Durch Drücken des Stopp-Knopfs während einer laufenden Berechnung wird diese abgebrochen.
Auto Calculate	Wenn die Option Automatisch berechnen aktiviert ist, werden alle Berechnungen unmittelbar nach einer Messung durchgeführt.
Runtime 2.0	Wählen Sie die Laufzeitumgebung aus. Für vorhandene, perfekt laufende Formeln empfehlen wir, diese wie konfiguriert beizubehalten (auch in Version 1.0). Wählen Sie für neue Formula Editor-Anwendungen die neueste aus.
Compile Formula	Erstellt eine DLL aus dem Code der Formel. Je nach ausgeführten Operationen, kann diese eine Leistungssteigerung von Faktor 50 oder mehr bedeuten. Zusätzlich bietet diese Funktion einen Schutz des Codes, da diese nicht mehr zu lesen ist.

4.9.2 Debug

Debug	Mit dem grünen Pfeil "Start Debug" wird die Formel bis zum ersten Break-Point ausgeführt, die Berechnungen werden an dieser Stelle dann angehalten und die aktuellen Resultate können bei Traces und Results entnommen werden
Next Breakpoint	Durch Klicken auf den Knopf "Next Breakpoint" können die Berechnungen weitergeführt werden.
Step Over	"Step Over": Wie beim Knopf "Next Step" wird die Berechnung einer Zeile ausgeführt, wobei jedoch Unterfunktionen übersprungen werden.
Next Step	Durch Drücken des Knopfs mit dem Pfeil abwärts "Next Step" kann anschliessend jede weitere Zeile der Formel einzeln abgearbeitet werden.
Step out	Mit dem Knopf "Step Out" werden die Berechnungen einer Schlaufe (for loop etc.) abgeschlossen. Es wird bei der nächsten Zeile nach der Schlaufe wieder gestoppt.

4.9.3 Controls

Function Browser	Schaltet das Fenster "Function Browser" zu. Aus diesem können alle Funktionen, Operatoren und Kanäle ausgewählt werden.
Traces	Schaltet das Fenster "Traces" zu. Die in den Formeln berechneten Kurven werden in diesem aufgelistet. Solche Resultat-Kurven können via Drag & Drop in ein Kurven-Anzeigefenster gezogen werden.
Results	Schaltet das Fenster "Results" zu. Die in den Formeln berechneten Skalare (Zahlenwerte, keine Kurven) werden in diesem aufgelistet.
Help	Schaltet das Fenster "Help" zu. Hier wird zu jeder verfügbaren Funktion eine kurze Beschreibung angezeigt, sobald diese im "Function Browser" angewählt ist.
Errors	Dieses Fenster zeigt zusätzliche Staus und Fehlermeldung des Formel Editor und kann zum Debuggen verwendet werden.

4.9.4 Einstellungen

Font Size 13	Die **Schriftgrösse** im Fenster **"Formula"** lässt sich individuell einstellen und wird mit den Layout-Einstellungen gespeichert.
Tab Size 4	Die Taste Tabulator wird durch die entsprechende Anzahl Leerschläge ersetzt.
☑ Auto Completion	Wenn diese Checkbox angewählt ist, werden während der Eingabe Vorschläge zur automatischen Vervollständigung eingeblendet. Mit den Tasten Pfeil auf- und abwärts kann der passende Vorschlag ausgewählt und mit der Taste "TAB" oder Leerschlag übernommen werden. Mit der Taste "ESC" wird die Vervollständigung abgebrochen.
☐ Auto Formatting	Formatiert den Code gemäss den Angaben von "Tab Size". So werden z.B. verschachtelte for loops wieder richtig formatiert.

4.9.5 Formula Editor Controls

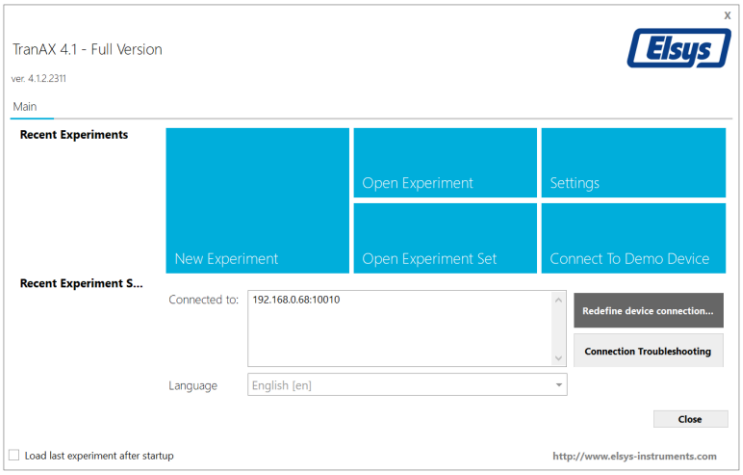
Formula Page	Öffnet Eine neue Formeleditor Seite.
Add Formula	Fügt der Seite einen neuen Formel Editor hinzu.

5 Erste Schritte

Dieses Kapitel gibt eine Einführung in die fast unbegrenzten Aufnahme- und Auswerte-Möglichkeiten von TranAX. Für Anwendungen, bei welchen auf die speziellen Möglichkeiten verzichtet werden kann, verweisen wir auf die sogenannte SCOPE-Anwendung.

5.1 Startseite

Beim erstmaligen Starten von TranAX erscheint zu Beginn eine **Startseite**. Hier werden die kürzlich geladenen Experimente sowie Experiment Sets auf der linken Seite abgebildet.



Auf der rechten Seite findet man Buttons zum Erstellen und Öffnen von einem Experiment sowie das Öffnen von Experiment Sets.
Neu kann man auch direkt auf ein **Demo Device** verbinden, wenn man keine Hardware zur Verfügung hat. Die Startseite erscheint bei jedem Neustart von TranAX.

Falls man jedoch lieber das letzte Experiment geladen haben möchte, kann man in der linken unteren Ecke die Checkbox **„Letztes Experiment nach dem Starten laden“** anhaken. Im Menu *"Datei / Startseite öffnen"* kann dieser Dialog jederzeit wieder angezeigt werden.

Falls keine Messkarten im System verbaut sind, kann durch Drücken der Schaltfläche **"Demo Device verbinden"** ein Messsystem mit simulierten Kurven aktiviert werden. Um diese Kurven zu nutzen muss im Control Panel erst auf das Lokale Gerät verbunden werden.

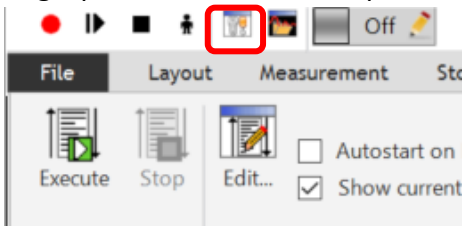
Ebenfalls beim Start erscheint ein Quicktipp Dialog, welcher auf praktische Funktionen und Möglichkeiten hinweist. Dieser kann jederzeit in den Settings von TranAX deaktiviert oder aktiviert werden.

Als nächsten Schritt kann man entweder ein neues Experiment erstellen oder ein bestehendes auswählen. Es sind bereits vordefinierte Beispiel Experimente vorhanden falls man TranAX mit dem Setup installiert hat.

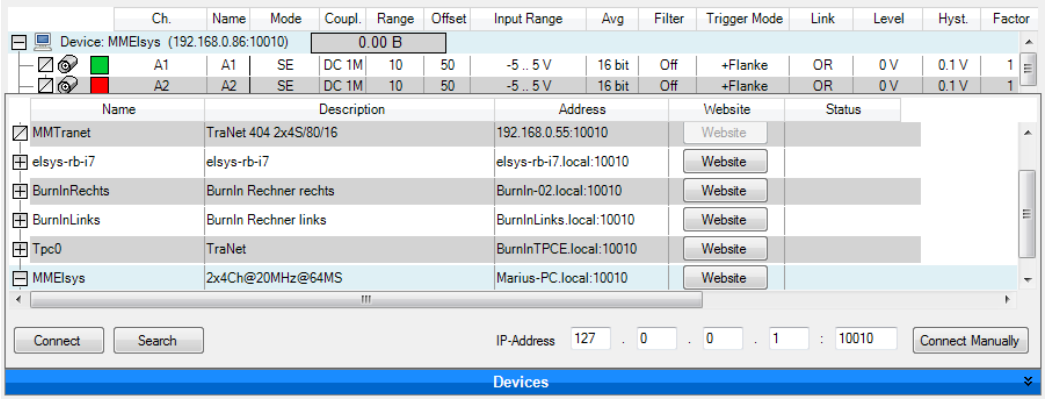
Um ein bestehendes Experiment (TranAX 3) zu laden, muss im Dialog " Öffnen" die **Dateiendung von *.exp auf *.lay oder *.all** umgestellt werden.

5.2 Device Manager

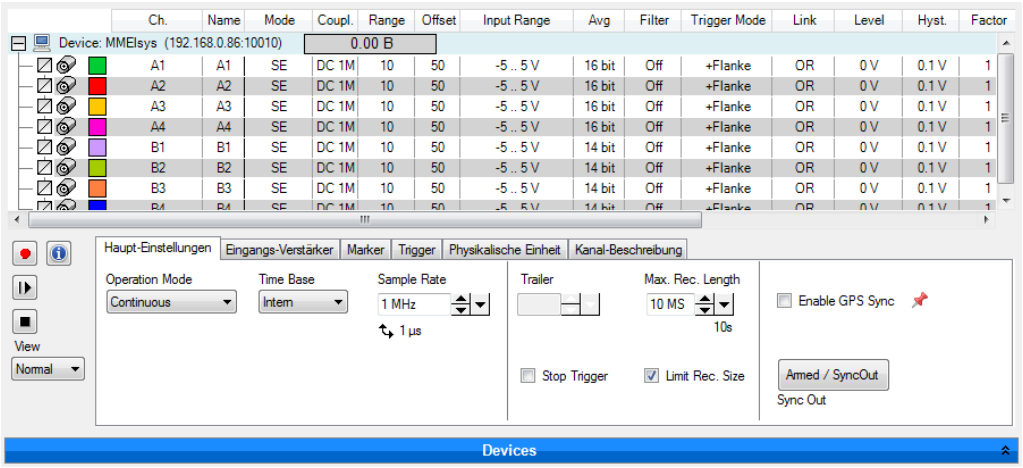
Um die eingebauten TPCX/TPCE-Module oder externe Geräte wie TraNET FE zu benutzen, muss TranAX mit diesen verbunden werden. Da die gesamte Kommunikation TCP/IP basierend ist, werden diese, wie in einem Netzwerk üblich, über deren IP-Adresse und dem entsprechenden Port angesprochen. Wenn ein Experiment geladen wurde, kann man das Control Panel öffnen:



Im Control Panel werden alle verbundenen Geräte aufgelistet. Um auf ein Gerät zu verbinden klickt man ganz unten auf den Devices-Expander. Der Device Manager erscheint in einem überlappenden Fenster. Hier werden nun alle Geräte aufgelistet, die im Netzwerk ersichtlich sind:



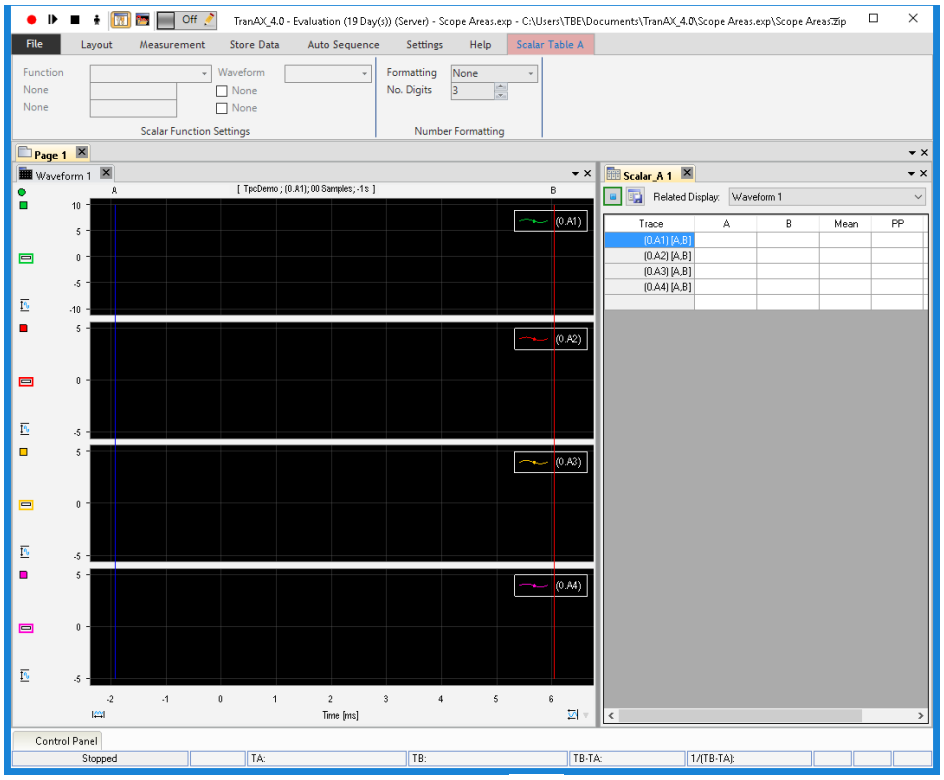
Man selektiert mit der Maus sein gewünschtes Gerät und drückt auf „Connect“. Im Control Panel erscheint nun das Gerät:



5.3 Einfaches Experiment: Template "Scope Areas"

Klicken Sie auf der Startseite auf "Neues Experiment" und laden Sie die Experiment Vorlage (Template) "Scope Areas". Der Name und das Verzeichnis für das neue Experiment wird automatisch vorgeschlagen und kann vor dem Erstellen noch angepasst werden. Experimente können zu einem späteren Zeitpunkt verschoben und umbenannt werden.

Im Verzeichnis "C:\ProgramData\Elsys\TranAX_4.0\ExperimentTemplates" sind die Vorlagen abgelegt. Diese dienen als Basis für eigene Experimente.
Eigene erstellte Vorlagen (Experiment Sets) können in dieses Verzeichnis kopiert als weiteres Template verwendet werden.



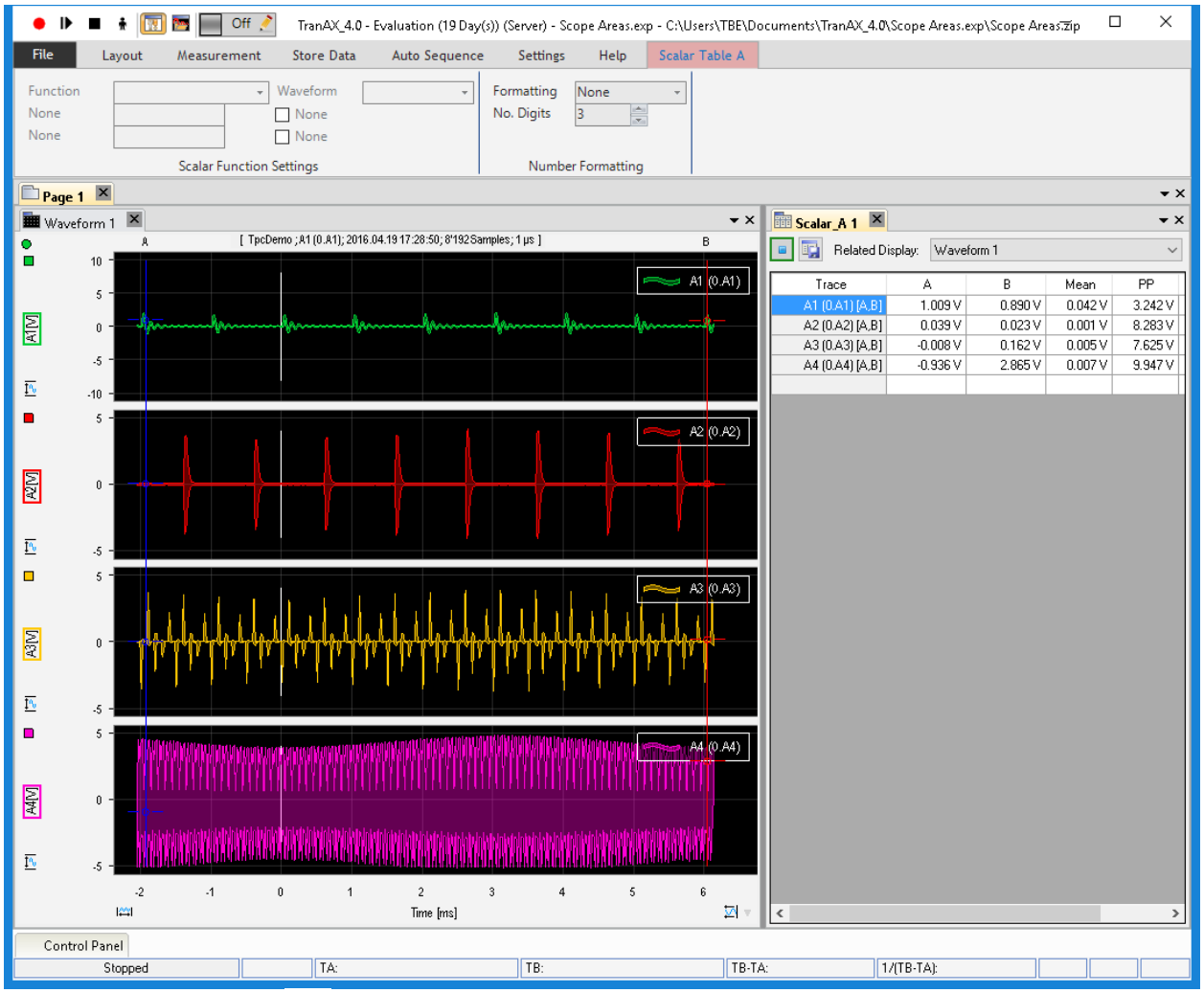
Als nächstes muss bei "Control Panel"  bei "Devices" das angeschlossene Gerät oder die installierten Messmodule verbunden werden.

Sollte keine Hardware installiert sein, kann auf der Startseite auch direkt mit dem "Demo Device" verbunden werden. Ist diese Schaltfläche orange hinterlegt, ist die Simulierte Hardware bereit und verbunden.

Der Demo Server wird nicht bei den Devices aufgelistet! Soll manuell auf diesen verbunden werden so ist zu beachten, dass beim Port 10030 eingestellt ist.
Connected to: 127.0.0.1:10030 (Demo Device)

5.3.1 Messung starten

Durch Klicken auf den roten **Start-Knopf** ● oder drücken von F6 wird die Messung gestartet. Im Waveform sind nun die Messkuren zu sehen, Die Skalar Tabelle rechts davon zeigen die berechnet Werte anhand der beiden Cursors A und B an.



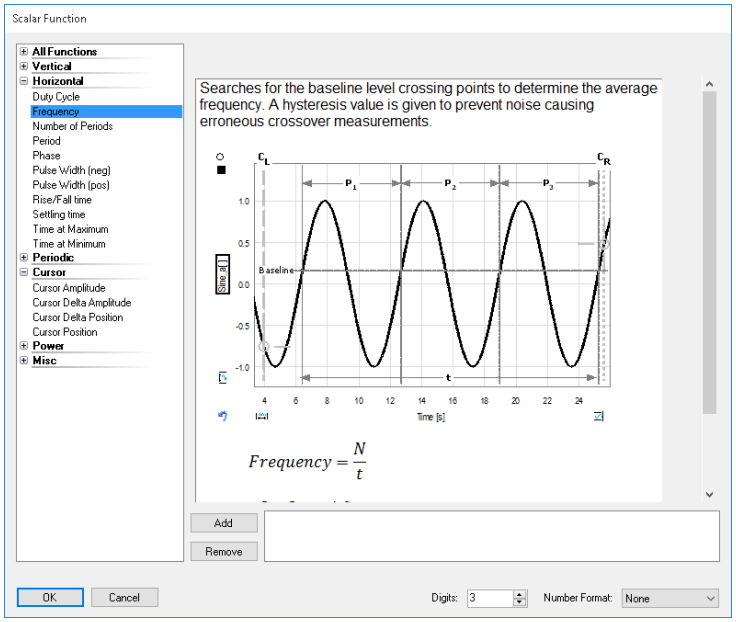
Mit dem Stopp-Knopf ■ oder drücken von F8 wird die Aufnahme beendet. Für weitere Möglichkeiten und Einstellungen kann im Control-Panel die einzelnen Kanäle angepasst werden.

5.3.2 Anpassen der Skalar Tabelle

Sollen andere Werte in der Skalar Tabelle dargestellt werden, können entweder neue Spalten eingefügt oder bestehende ersetzt werden. Durch Doppelklick auf das oberste Feld der Spalten können deren Eigenschaften angepasst werden.

Soll anstelle der Werte an der Position des Cursors A die Frequenz der einzelnen Signale gemessen werden kann die Spalte A Doppel geklickt werden und der Skalar Funktionen Dialog erscheint.

Trace	A	B	Mean	PP
A1 (0.A1) [A,B]	1.009 V	0.890 V	0.042 V	3.242 V
A2 (0.A2) [A,B]	0.039 V	0.023 V	0.001 V	8.283 V



Hier kann bei den Horizontalen Funktonen "Frequency" gewählt werden und anschliessend unten links den Knopf "OK" drücken.

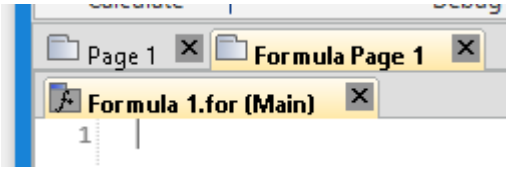
Anschliessend werden die gemessenen Frequenzen der Signale dargestellt.

Trace	Freq	B	Mean	PP
A1 (0.A1) [A,B]	8.027 kHz	0.890 V	0.042 V	3.242 V
A2 (0.A2) [A,B]	7.921 kHz	0.023 V	0.001 V	8.283 V
A3 (0.A3) [A,B]	6.016 kHz	0.162 V	0.005 V	7.625 V
A4 (0.A4) [A,B]	60.000 kHz	2.865 V	0.007 V	9.947 V

Der analysierte Bereich kann durch schieben der Cursors (hier A und B) angepasst werden.

5.3.3 Analyse mit Formeln

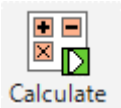
Mit Hilfe des Formel Editors können auch komplexe Analysen der Kurven vorgenommen werden. Im folgenden Beispiel wird ein Stück einer Kurve ausgeschnitten, gefiltert und wieder im Waveform angezeigt. Zusätzlich wird noch ein Skalar Wert "Max" in der Formel berechnet.



Im Ribbon Tab "Layout" bei der Gruppe "Formel Editor Controls" den Knopf "Formel Seite" klicken und ein neues Formel Fenster öffnet sich.

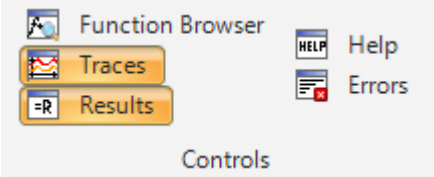
Formel für dieses Beispiel:

```
; Don't create files for
; calculated traces
; Will be faster!
UseMemory(True)

; Copy channel A1 to the variable "traceOrig"
traceOrig = c0A1

; --- slice out section between cursors ---
tA = GetCrs("Waveform 1","A")
tB = GetCrs("Waveform 1","B")
traceSlice = Slice(traceOrig,tA,tB)

; --- filter sliced trace ---
; Lowpass filter:
; Bessel, 6th order, cutoff frequency at 10kHz
traceFilter = LowPass(traceSlice,Bessel,6,10E3)

; --- calculate some scalar values---
maxOrig = Max(traceOrig)
maxSlice = Max(traceSlice)
maxFilter = Max(traceFilter)
```

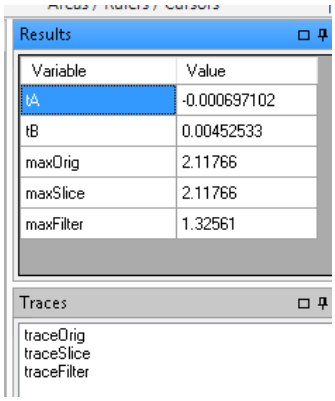


Um die Formel zu berechnen, kann in der Ribbon Bar "Formel Editor", Gruppe "Berechnung" der Knopf "Berechnen" oder die Taste F10 gedrückt werden.

Die Taste F10 führt die Berechnung der Formel aus, auch wenn der Formal Tab nicht aktiv oder der Knopf "Berechnen" nicht sichtbar ist.



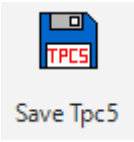
Um die berechneten Variablen zu sehen, kann in der Ribbon Bar "Formel Editor", Gruppe "Controls" die Punkte "Kurven" und "Resultate" aktiviert werden.

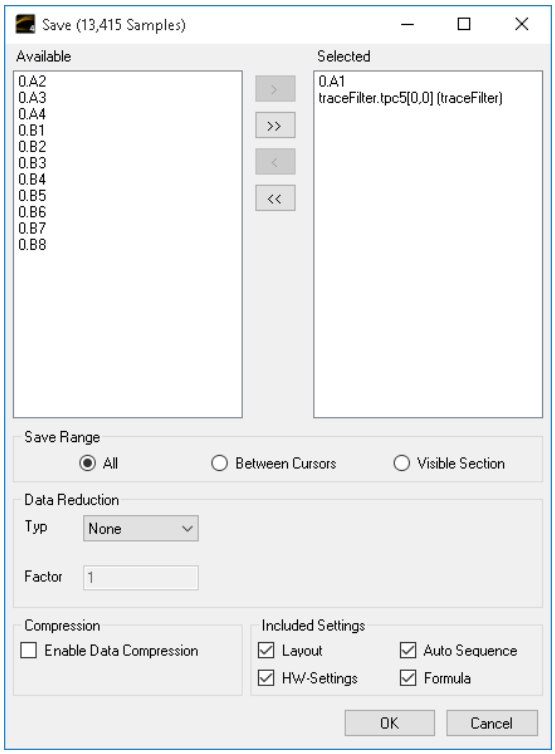


Bei "Traces" sind die berechneten Kurven, bei "Results" die Berechneten Skalar Werte gelistet.
Die Kurven bei "Traces" können via Drag & Drop in ein Waveform gezogen werden um diese zu Visualisieren.

5.3.4 Aufnahmen speichern und laden

Nachdem die Aufnahmen beendet und die Kurven berechnet sind, können Diese für spätere Analyse oder Archivierung gespeichert werden. In der Ribbon Bar "Daten speichern", Gruppe "Daten speichern und laden" kann der Knopf "Tpc5 speichern" gedrückt werden.





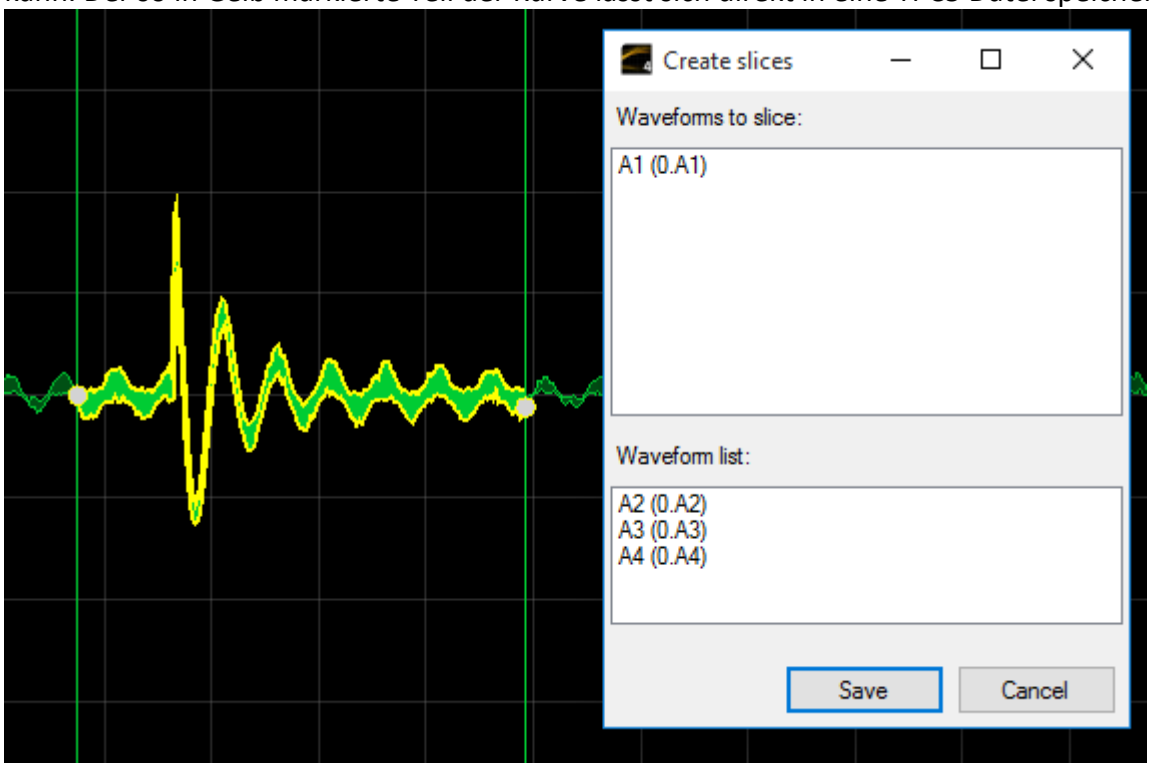
In der linken Liste sind die verfügbaren Kurven gelistet, auf der rechten die welche gespeichert werden sollen. Durch Doppelklick auf eine Kurve oder durch Klicken der Schaltfläche werden diese hinzugefügt. Berechnete Kurven müssen bereits in ein Waveform gezogen worden sein, damit diese gelistet werden. Durch klicken von "OK" wird der Speichern Dialog geöffnet und die Kurven können gespeichert werden.

Für mehr Informationen über die Speicher Optionen, gehen Sie zum Abschnitt "Kurven speichern".

Bereits gespeicherte Signale können wieder geladen und analysiert werden. Das Signal einer TPC5-Datei kann mit dem Signalquellen-Browser geöffnet werden.

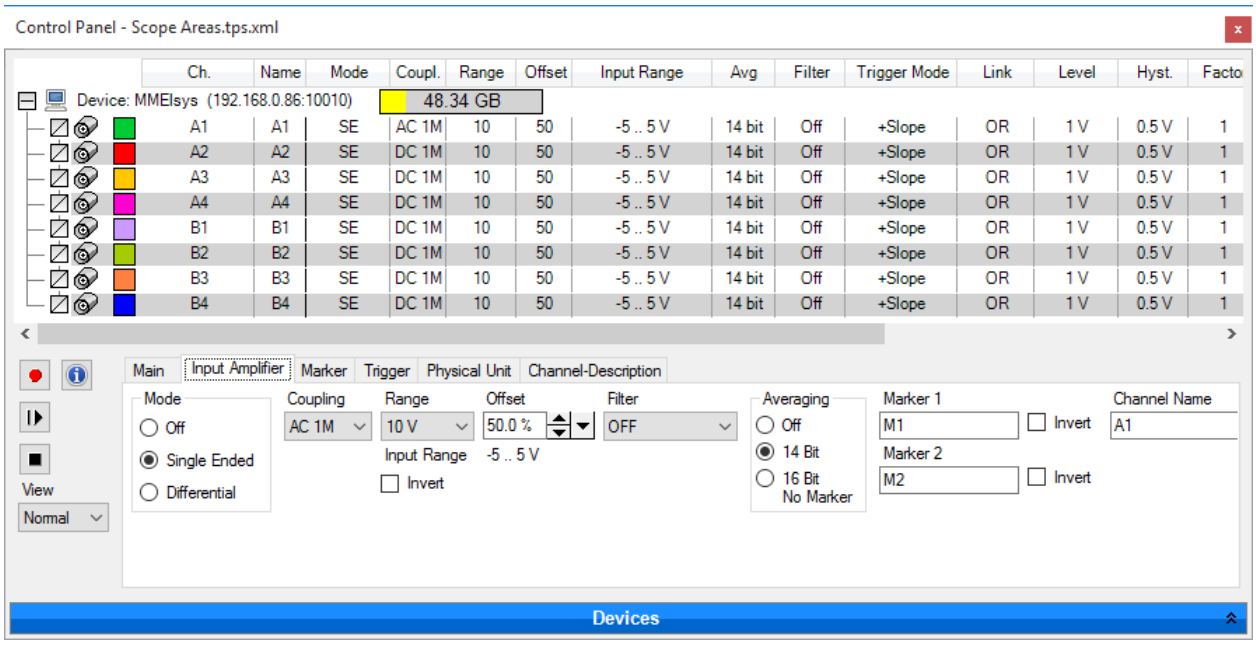
5.3.5 Ausschnitt einer Kurve Speichern

Durch einen Rechts-Klick auf eine Messkurve lässt sich im Popup Menü die Funktion "Create Slices" auswählen. Es erscheint ein neuer Cursor, welcher nach Links oder Rechts gezogen werden kann. Der so in Gelb markierte Teil der Kurve lässt sich direkt in eine TPC5 Datei speichern.



6 Control Panel

Das Control Panel wird verwendet um **Aufzeichnungs-Parameter** wie Abtastrate, Eingangsverstärkung etc. einzustellen. Das Fenster ist in **zwei Bereiche** aufgeteilt:



Der **obere Bereich** enthält eine Liste mit allen vorhandenen Kanälen und deren Einstellungen. Die Kanäle werden beim Aufstarten automatisch erkannt. Man kann einen oder mehrere Kanäle auswählen um sie zu modifizieren. Falls mit gehaltener Maustaste über die Kanäle gefahren wird, werden diese markiert.
Die Signal-Kurven können von den **ersten beiden Kolonnen** in der Kanal-Liste in die Kurvenanzeige gezogen werden (**Drag & Drop**). Daher funktioniert das Markieren mit gehaltener Maustaste nicht für diese ersten beiden Kolonnen.

Die Liste im oberen Bereich des Control-Panels kann zu Dokumentations-Zwecken in eine Text-Datei gespeichert werden. Dazu sind die Kanal-Linien die gespeichert werden sollen zu markieren. Danach kann via <Strg>-S der Speichervorgang eingeleitet werden.

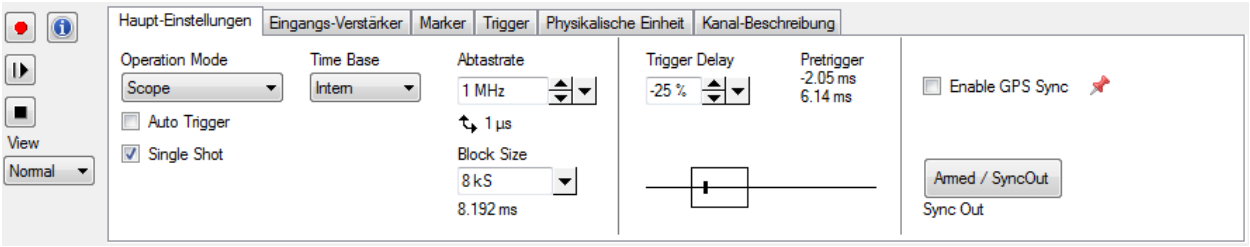
Im **unteren Bereich** des Control Panels werden die jeweiligen Einstellungen vorgenommen. Die verschiedenen Parameter sind in Registern nach Parameter-Klassen unterteilt.

6.1 Haupt-Einstellungen

Mit dem TranAX Datenerfassungs-System lassen sich sehr schnelle Signale (Transienten), aber auch lang andauernde, vereinzelt oder periodisch auftretende Signale digitalisiert erfassen und abspeichern.

Die Hardware-Grundausführung besteht aus einem **Datenerfassungs-Modul mit 4 oder 8 Kanälen**. Je nach Platz im Rechner können zusätzliche Erweiterungsmodule eingebaut werden. Konzeptionell ist ein Ausbau von bis zu **64 Kanälen** in einem Instrument möglich.

Die Signale können in mehreren Kanälen parallel aufgezeichnet werden, wobei jeder Kanal seinen eigenen Speicherteil besitzt, unabhängig vom Arbeitsspeicher des Computers. Die Trigger Bedingung, d.h. die Voraussetzungen für das Auslösen bzw. Abschliessen einer Datenaufnahme, kann für verschiedene Kanäle individuell vorgegeben werden.
Die Datenerfassungs-Hardware, TPCX, TPCE oder TraNET, kennt grundsätzlich drei verschiedene Aufnahme-Betriebsarten.



Die Registerkarte ***Haupt-Einstellungen*** enthält folgende einstellbaren Parameter:

- Betriebsart: Die Datenerfassungs-Hardware, TPCX, TPCE oder TraNET, kennt grundsätzlich die folgenden Aufnahme-Betriebsarten.
    - Scope (mit Autotrigger und/oder Single Shot)
    - Multiblock
    - Kontinuierlich
    - ECR (Event Controlled Recording)
- Zeitbasis: Intern oder Extern
- Die Abtastrate (interne Zeitbasis) oder erwartete Taktfrequenz (externe Zeitbasis, wird für gewisse mathematische zeitrelevante Funktionen verwendet)
- Aufzeichnungslänge (Blocklänge mit Pre- und Post-Trigger)
- Trigger Verzögerung (legt Pre- und Post-Trigger Verhältnis fest)

Mit dem Knopf ***"Armed / SyncOut"*** kann der entsprechende Ausgang am Digital-Stecker als Armed-Signal oder als Ausgang von Takt-Pulsen, mit einstellbarer Frequenz, geschaltet werden. Diese Frequenz ist unabhängig von der eingestellten Zeitbasis. Sie dient zur Synchronisation von externen Geräten (z.B. Highspeed-Video-Kameras).
Bei älteren Geräten ist dieser Knopf nicht vorhanden. Die Hardware dieser Geräte müsste im Lieferwerk nachgerüstet werden.

Die **Abtastrate** kann entweder als **Abtastfrequenz** oder als **Zeitintervall** angezeigt und eingegeben werden. Das Eingabefeld reagiert auf einen beigefügten Dezimalvorsatz ("m" für Milli, "k" für Kilo, "M" für Mega, "u" für Micro) und eine Einheit ("H" oder "Hz" für Hertz, "s" für Sekunden). Beispiele:

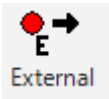
Eingabe	Interpretiert als
10**k**	10 **kHz**
2.5**M**	2.5 **MHz**
2.5**m**	2.5 **ms**

Eingabe	Interpretiert als
15**u**	15 **µs**
500**Hz**	500 **Hz**
3**s**	3 **sec**

Die Aufnahmezeit (Zeitfenster) wird von der Blocklänge und der Abtastrate bestimmt.

6.2 Icons

Die Schaltflächen links im Control Panel haben folgende Funktionen:

	Aufzeichnung starten (F6).  Alternativ dazu kann die Datenerfassung gestartet werden mit einem externen TTL-Signal, das mit der Schaltfläche "External" im Ribbon Tab "Messung", Gruppe "Messung" freigeschalten wird.
	Manueller Trigger Modus (F7)
	Aufzeichnung manuell stoppen (F8)
	Hardware Konfiguration anzeigen
View Normal	Hier kann zwischen Normal- und Clusteransicht umgeschaltet werden. In der Clusteransicht können einzelne Boards einem Cluster zugewiesen werden um für den jeweiligen Cluster einen anderen Messmodus zu wählen.

6.3 Control Panel Register

Um Einstellungen zu verändern, verwenden Sie die Dropdown Menüs und Optionen-Listen. Textfelder können entweder manuell ausgefüllt werden oder mittels folgender Schaltflächen:

	Wert erhöhen oder verringern
	Werte aus der Liste auswählen
	Wechselt die Einheit der Parameter (z.B. Frequenz/Abtastintervall)

Einige Hardware-Einstellungen werden in kleinen **Illustrationen** bei den Werten dargestellt. Sie können dann die Einstellungen auch **mittels der Markierungen ändern,** wie z.B. die Trigger Verzögerung.

7 Virtual Hardware Manager (VHM)

7.1 Kurzbeschreibung des VHM

Elsys Virtueller Hardware Manager (kurz VHM) bietet neue Möglichkeiten für die Anwendung und Nutzung der TraNET Geräte, TPCX und TPCE Karten, zum Messen, Aufzeichnen und Analysieren der Messdaten.

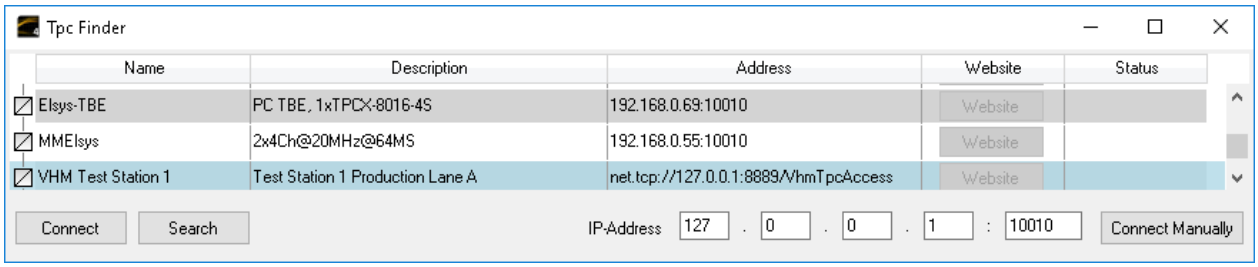
Die gegebenen Limitationen zwischen Software Konfiguration und Hardware-Setup kann mit VHM ignoriert werden. Die Erstellung von Experimenten und Experiment Sets ohne vorhandene Hardware sowie die Kombination und verschiedener Geräte und unterschiedlicher Kanalanzahl ist sehr nun einfach möglich.

Einige Vorzüge und Merkmale:

- Server Anwendung (VHM Server), handhabt die Geräte und DAQ Module
- Client Anwendung (VHM Client), einfache Konfiguration von virtuellen Servern auf unterschiedlichen Systemen im selben Netzwerksegment
- Erstellen von virtuellen Geräten
- Erstellen von virtuellen Modulen, verbinden von 2x4 Kanälen zu 1x8 Kanälen
- Einfache Handhabung, Drag & Drop vorhandene DAQ Module in ein virtuelles Device
- Sperrfunktionalität und Passwortschutz der Server Anwendung

7.2 Ein VHM Device verbinden

Virtual Hardware Manager (VHM) Devices werden im Device Manager wie andre physikalisch vorhanden TraNET Geräte aufgelistet. Anstelle des TraNET Gerät wird das VHM Device ausgewählt und damit verbunden.



7.3 Systemanforderungen

Um den Virtuellen Hardware Manager und dessen Vorteile nutzen zu können, sind die folgenden Software Versionen notwendig:

Software	Version
Virtual Hardware Manager	4.1.0.1795
TranAX	4.1.0.1795

Detailliertere Informationen zum Elsys Virtual Hardware Manager (VHM) sind im separaten Handbuch zu finden. Bei Fragen kontaktieren Sie Ihren lokalen Vertreter oder direkt Elsys Instruments: info@elsys-instruments.com

8 Aufnahme-Betriebsarten

8.1 Scope

Der Scope-Mode ist der Grundeinstellungs-Mode. Es werden keine Daten auf die Festplatte gespeichert. Die aktuelle Aufzeichnung ist nur im Speicher der TPCX/TPCE-Module vorhanden. Nachdem die Aufzeichnung gestartet und abgeschlossen wurde, wird der aktuelle Block angezeigt.
In dieser Registerkarte kann man die Zeitbasis, Abtastrate, Blocklänge und Trigger Verzögerung eingeben. Die Illustration unterhalb der Trigger Verzögerung zeigt den Trigger Punkt grafisch in Relation zu der gesamten Aufzeichnungslänge. Die Trigger Verzögerung lässt sich auch grafisch mit der Maus einstellen.

☑ Auto Trigger	Falls kein Trigger-Ereignis eintritt, triggert TranAX automatisch.
☑ Single Shot	Nur ein einziges Ereignis wird angezeigt und die Aufzeichnung beendet.

Falls mehr als nur ein Cluster konfiguriert ist, ist die Checkbox **Single Shot** immer aktiviert (Auto-Start Aufnahme ist nicht möglich).
Alternativ kann eine Auto Sequenz mit dem Kommando "Start Recording" in einer Schlaufe verwendet werden.

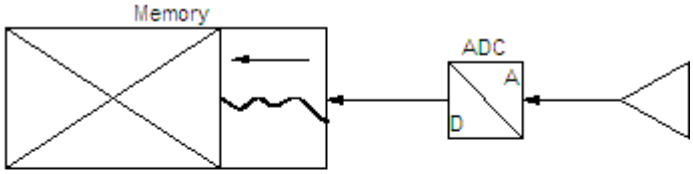
8.2 Multiblock

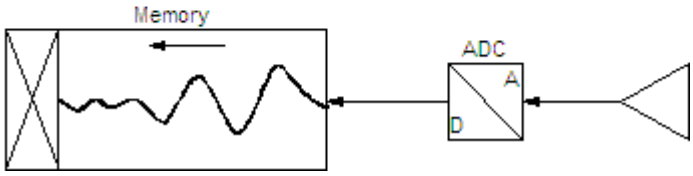
Der Multiblock-Mode speichert die aufgezeichneten **Signale fortlaufend** in Segmente des Kanal-Speichers. **Jedes Trigger-Ereignis wird in einem Block gespeichert**. In dieser Registerkarte können auch die **Zeitbasis**, die **Abtastrate**, die **Blocklänge** und die Trigger Verzögerung eingestellt werden. Zusätzlich zum Scope-Mode können hier die Anzahl Blöcke bzw. Anzahl der Aufzeichnungen festgelegt werden. Die maximale Anzahl an Blöcken hängt von der Blocklänge und vom gesamten Modul-Speicher ab.

Der Multiblock-Mode ist speziell geeignet für das Erfassen von Bursts mit einer **hohen Repetitionsrate** und **minimaler Totzeit** zwischen den Bursts. Falls die Anwendung eine totzeitlose Erfassung von Bursts verlangt, so ist der ECR-Mode der ideale Datenerfassungs-Mode.

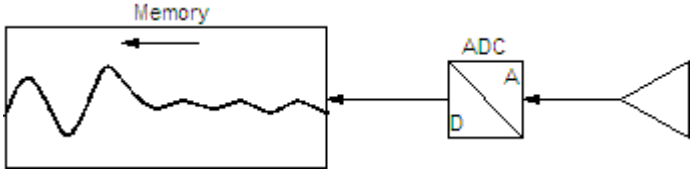
Im Multiblock-Mode übernehmen die TPCX/TPCE-Module oder das TraNET-Messgerät vollständig die Kontrolle über den Ablauf und die Speicherung einer Messung. Die Messung hat eine vorher **festgelegte Speicherlänge (Blockgrösse)**, die in den auf den Modulen verfügbaren Arbeitsspeicher passen muss. Der Rechner ist an der Messung nicht beteiligt, er gibt nur das Start-Kommando und wartet bis die Messung beendet wurde. Danach liest TranAX die Daten aus. Im Folgenden wird das Prinzip einer Messung im Block-Mode beschrieben.

Die TPCX/TPCE-Module beginnen sofort nach dem **Start der Messung** damit, ihre Speicher mit **Werten von den ADCs** zu füllen, bis die eingestellte **Blockgrösse erreicht** wurde.

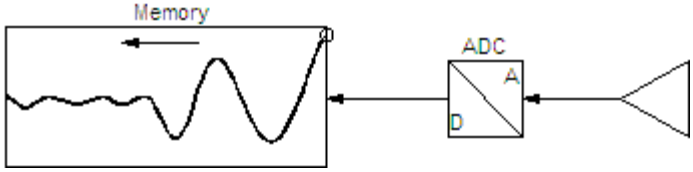




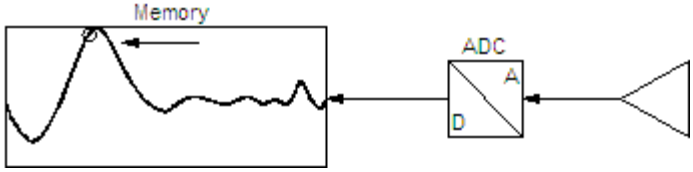
Von jetzt an werden die **ältesten Werte** fortlaufend durch die neuen **überschrieben**.



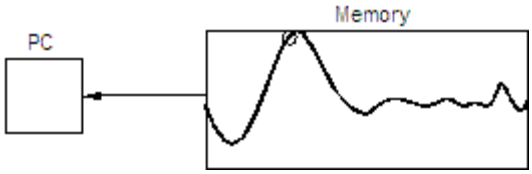
Durch dieses Verfahren liegt immer das letzte Stück des Signals aus der Vergangenheit bis zur **Gegenwart** im Speicher. Tritt ein **Trigger-Ereignis** ein, kann die Aufnahme angehalten werden, und man hat den Signalverlauf **vor dem Trigger Punkt**.



Falls man am weiteren Signalverlauf **nach dem Trigger Punkt** interessiert ist, kann die TPCX/TPCE-Hardware noch eine Weile weiteraufzeichnen, bevor die Aufnahme beendet wird. Siehe auch Pre- und Post-Trigger (Triggerverzögerung).



Nachdem die Aufnahme beendet wurde, werden die aufgenommenen Daten aus dem Modulspeicher an den PC übertragen.



8.3 Kontinuierlich

Im kontinuierlichen Mode arbeitet das DAQ Device oder Transientenrekorder als **Disk-Rekorder**. Es können keine Blocklängen oder Pre/Post Trigger-Einstellungen gemacht werden.

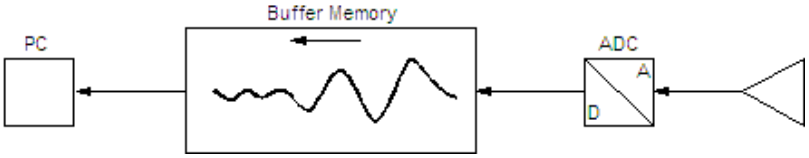
Die Aufzeichnung wird mittels der Start-Schaltfläche gestartet. Um die Aufzeichnung zu beenden gibt es drei Möglichkeiten:

Trailer 1 kS 1ms Stop Trigger	Die **Aufzeichnung wird beendet**, falls ein Trigger-Ereignis auftritt. Zusätzlich kann eine Nachlaufzeit festgelegt werden. Die Aufzeichnung wird nach einem Trigger-Ereignis noch um eine voreingestellte Dauer fortgesetzt. Die Trigger-Einstellungen werden in der Trigger Registerkarte eingegeben.
Max. Rec. Length 10 MS 10s Limit Rec. Size	Die Aufzeichnung wird nach einer bestimmten **Zeitdauer automatisch beendet**, falls dies nicht durch einen allfälligen Stopp-Trigger vorher erfolgt ist. Die Aufzeichnungsdauer kann auch vom verbleibenden Platz auf der Festplatte abhängen.
■	Die Aufzeichnung kann manuell gestoppt werden.

Falls beide Optionen aktiviert sind, wird die Aufzeichnung beendet sobald eine der Bedingungen eingetreten ist.

Bitte beachten: Falls ein anderer Cluster einen Trigger auslöst, wird der Stopp Trigger auch im kontinuierlichen Mode ausgelöst!

Im kontinuierlichen Mode werden alle **aufgezeichneten Werte** sofort an den Computer übertragen. Dieser speichert sie auf die **Festplatte**. Die Aufzeichnung kann vom Computer aus oder durch die TPCX/TPCE-Hardware beendet werden, wenn diese ein Trigger-Ereignis erkennt. Die **Modul-Speicher** dienen in diesem Mode als **Buffer** für die gemessenen Werte.



Die **Aufnahmelänge** ist praktisch nur durch die **Festplatte begrenzt**. Durch die begrenzte Daten-Transferrate zwischen den Modulen und dem Computer sind jedoch der Abtastrate Grenzen gesetzt. Je nach Rechner kann die maximale **Summenabtastrate** bis zu einigen Dutzend Mega Samples pro Sekunde betragen. Diese **hohe Aufzeichnungsrate** wird erreicht, wenn sichergestellt ist, dass der Computer nicht durch fremde Programme oder Prozesse belastet wird. Bei reduzierter Abtastrate ist jedoch auch bei sporadischer Überlastung der CPU eine lückenlose Aufzeichnung sichergestellt, weil die Messwerte zuerst im Modul-Speicher gepuffert werden. Dieser **Aufzeichnungsmode** ist hauptsächlich für **lang andauernde, lückenlose Messungen** interessant.

Falls die Festplatte voll wird, wird die Aufzeichnung automatisch abgebrochen.

8.4 ECR Mode

Der ECR Mode ist optional erhältlich.

Der ECR Aufzeichnungs-Mode ermöglicht die gezielte Erfassung von **zyklisch** oder **sporadisch** auftretenden **Ereignissen**. D.h. die Registrierung von Messdaten erfolgt nur dann, wenn gewisse **Signal-Bedingungen** (Trigger, Zeitfenster, Wiederholungen usw.) erfüllt sind. Dadurch werden nur benötigte Signaldaten gespeichert.
Im ECR-Mode kann garantiert werden, dass keine Totzeiten auftreten und somit keine Ereignisse oder Information verloren gehen. Dies gilt auch, wenn sehr viele Kanäle mit maximaler Abtastrate über eine beliebig lange Zeit überwacht werden müssen. Da jeder Kanal seinen eigenen Signalpuffer (bis zu 64 MSamples) besitzt, darf lediglich die mittlere Anzahl der Ereignisse pro Sekunde einen gewissen Wert nicht überschreiten. Dieser Wert hängt von der einstellbaren Blocklänge pro Ereignis ab und ist zudem durch den max. möglichen Datendurchsatz direkt auf die Festplatte bestimmt (**je nach CPU/Disk-System ca. 20 MSamples pro Sekunde**). Die Trigger Bedingungen können für jeden Kanal individuell eingestellt werden, wobei sich auch komplexe Kriterien des Signalverlaufs definieren lassen (Advanced Trigger Option: Pulsbreite/Höhe, Periode, Steilheit, Fenster-Ein/Austritt usw.).

Im Gegensatz zum Block-Mode sind **Null-Totzeiten garantiert**. Bitte beachten Sie, dass im Block-Mode ein Trigger-Ereignis nicht aufgezeichnet wird, falls das Ereignis genau am Ende des Blocks auftritt. Der Bereich der Überlappung hängt von der Ereignis-Wiederholrate ab und kann in gewissen Grenzen mit Holdoff oder Retrigger gesteuert werden.

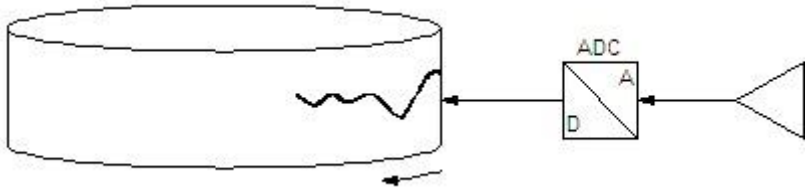
In der ECR-Betriebsart werden die anliegenden **Signalkurven laufend** in den als **Ringbuffer** geschalteten Modulspeicher geschrieben. Danach markiert jedes **Trigger-Ereignis** den **Adressbereich**, woraus die Signaldaten aus dem Modul-Speicher ausgelesen und in eine Datei geschrieben werden. Diese Aufzeichnungsart entspricht weitgehend dem bekannten Multiblock-Mode. Die **aufgezeichneten Zeitabschnitte** können sich hier jedoch **überlappen**. Nur dadurch ist eine Datenerfassung ohne Totzeit gewährleistet (vor allem bei Aufnahmen mit Pre-Trigger).

Falls man im ECR Mode ist, wird eine ECR Registerkarte angefügt. Die Blocklänge wird explizit via Pre- und Post-Trigger-Einstellungen ermittelt. Wie im Multi Block Mode kann die maximale Anzahl an Blöcken festgelegt werden, die aufgezeichnet werden sollen. Weiterhin wird ein Retrigger (RT) Marker oder ein Holdoff (HO) Marker unterhalb der Einstellungen angezeigt, je nach gemachten Einstellungen in der ECR Registerkarte.
Es gibt zwei verschiedene ECR Modi: Einzelkanal und Multikanal. Beide Modi unterstützen die Option Dual Mode.

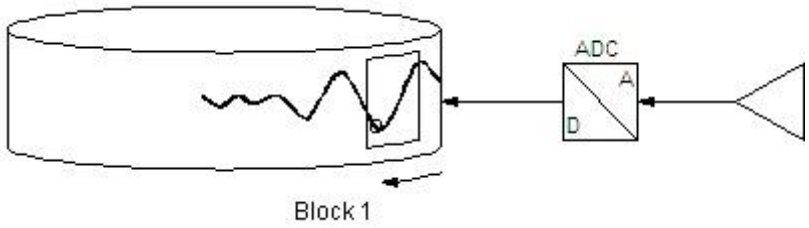
Falls die Trigger-Bedingungen etwas unkritisch gesetzt sind, kann es sein, dass die CPU von schnellen periodischen Signalen überbelastet wird. Das erweckt den Eindruck, als ob die CPU blockiert sei.

8.4.1 Prinzipieller Ablauf

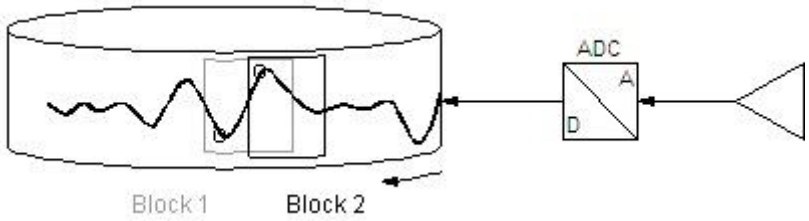
Der ECR-Mode läuft wie folgt: Das digitalisierte Signal wird auf dem internen Kartenspeicher (als **Ringbuffer** programmiert) gespeichert.



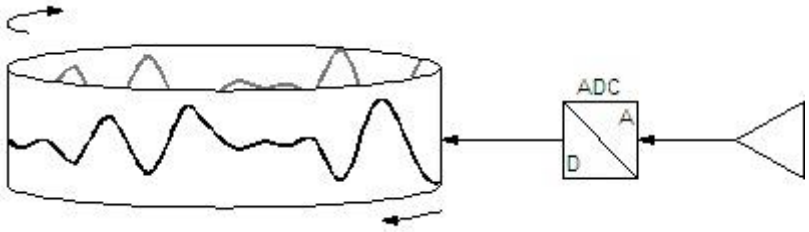
Sobald ein **Trigger-Ereignis** eintritt, werden die Block Samples vom Ringbuffer auf die **Festplatte gespeichert**.



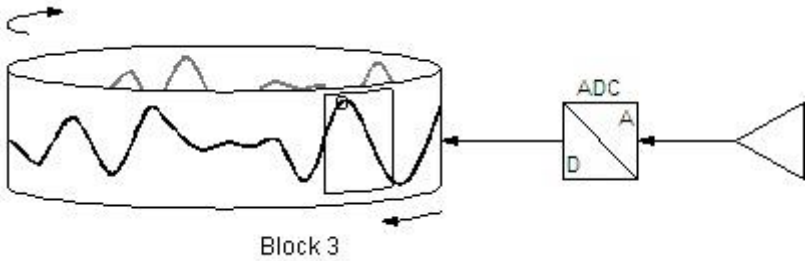
Falls ein **weiteres Trigger-Ereignis** während dem laufenden Block eintritt, wird ein überlappender Block aufgezeichnet.



Die **ältesten Aufzeichnungen** werden mit den neu **ankommenden Daten überschrieben**, sobald der Ringbuffer voll ist.
Normalerweise sind diese Daten **vorher** ausgelesen und auf die **Festplatte** transferiert worden. Falls zu viele Ereignisse gleichzeitig auftreten, kann es vorkommen, dass der **Ringbuffer überläuft**. TranAX zeigt dann eine entsprechende Statusmeldung an.



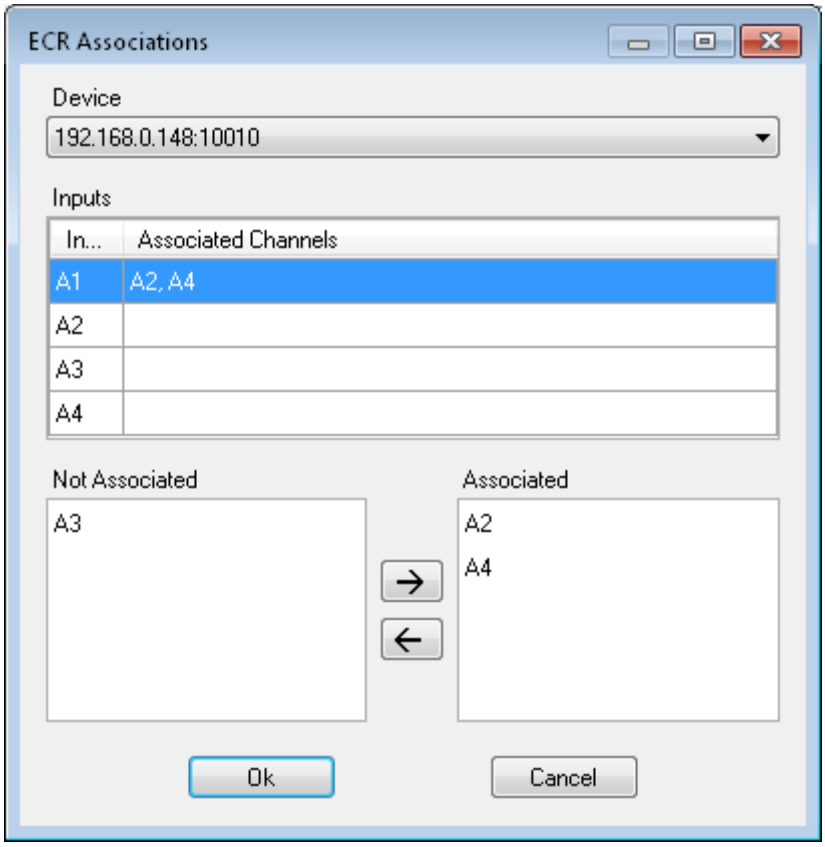
Nachdem die Anzahl eingestellter Blocks (in diesem Beispiel 3) erreicht wurde, wird die Aufzeichnung beendet.



8.4.2 ECR Single Channel Mode

Bei ECR Single Channel werden die Daten kanalbezogen registriert.

In diesem Mode kann **jeder Kanal als Einzel-Trigger-Quelle** gewählt werden. Es werden nur die Daten aus den entsprechenden Kanälen unabhängig voneinander registriert. Jedoch kann jedem **einzelnen Kanal assoziierte Kanäle zugeordnet werden**. Die Signale dieser assoziierten Kanäle werden dann im gleichen Zeitausschnitt zu den Signalen des triggernden Kanals abgelegt.
Durch Drücken auf **"ECR Assoziierung"** können Kanäle zugewiesen werden. Dabei erscheint folgendes Fenster:



Im oberen Teil in der Spalte "Eingang" kann der gewünschte Kanal gewählt werden. Danach können im unteren Feld mit dem rechten Pfeil → die zuzuweisenden Kanäle bestimmt werden.

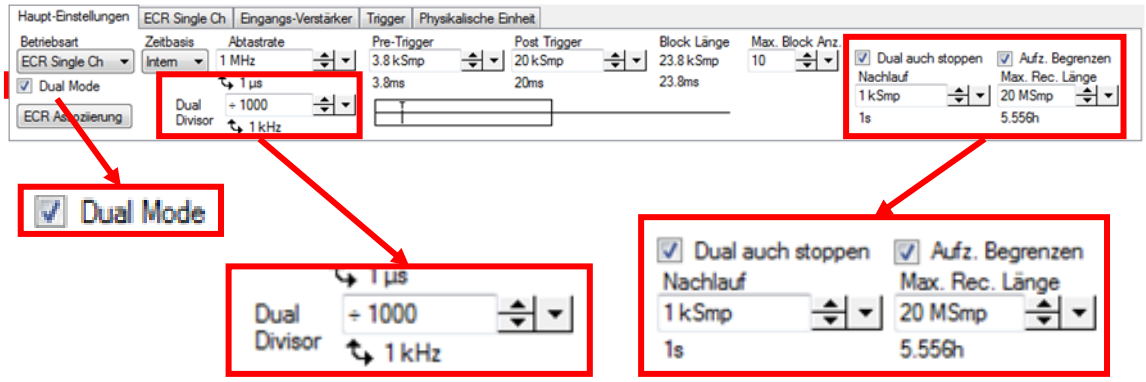
8.4.3 ECR Multi Channel Mode

Im ECR Multi Channel Mode werden die Signaldaten parallel aus allen eingeschalteten Kanälen aufgezeichnet.

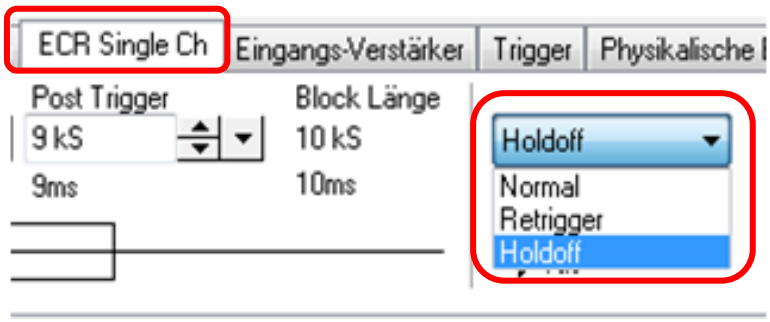
8.4.4 Dual Mode

Durch Zuschalten von **Dual Mode** wird der ECR-Mode **erweitert**. Dabei werden parallel zur Registrierung von schnell aufgezeichneten ECR Trigger-Ereignissen die **Signale von allen eingeschalteten Kanälen** kontinuierlich, **in der Regel relativ langsame Kurvenverläufe**, mit aufgenommen. Mit dem Parameter **Dual Divisor** lässt sich die Taktrate für die langsamere, kontinuierliche Aufzeichnung im Verhältnis zur schnellen Abtastrate einstellen.

Die kontinuierliche Aufzeichnung läuft bis auf die Start- und Stopp-Bedingung synchron zu den allenfalls auftretenden ECR-Ereignissen. Die langsame Aufzeichnung speichert immer die Signale aus allen eingeschalteten Kanälen in die Datei. Dies ist auch der Fall bei ECR-Single Channel. Es empfiehlt sich deshalb, nicht verwendete Kanäle auszuschalten.



8.5 ECR Trigger Optionen

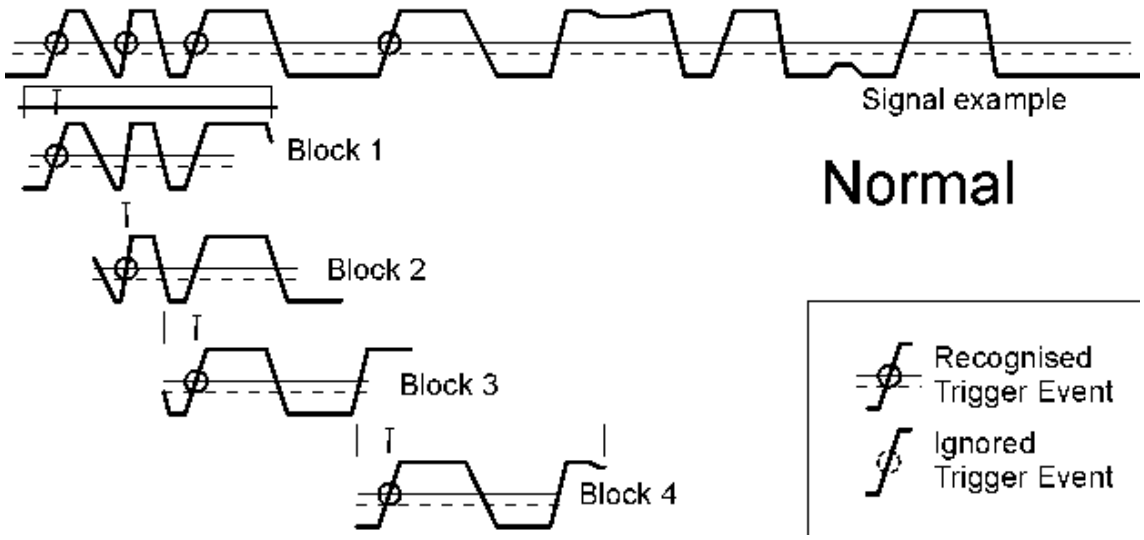


Zusätzlich zu den ECR-Einstellungen in der Registerkarte Haupt-Einstellungen, können in der Registerkarte ***"ECR Single Ch"*** beziehungsweise ***"ECR Multi Ch"*** die Optionen ***Normal***, ***Retrigger*** und ***Holdoff*** gewählt werden.

Pre-Trigger, *Post-Trigger* und *Anzahl Blöcke* können entweder unter *Haupt-Einstellungen* oder in der *ECR*-Registerkarte eingestellt werden.

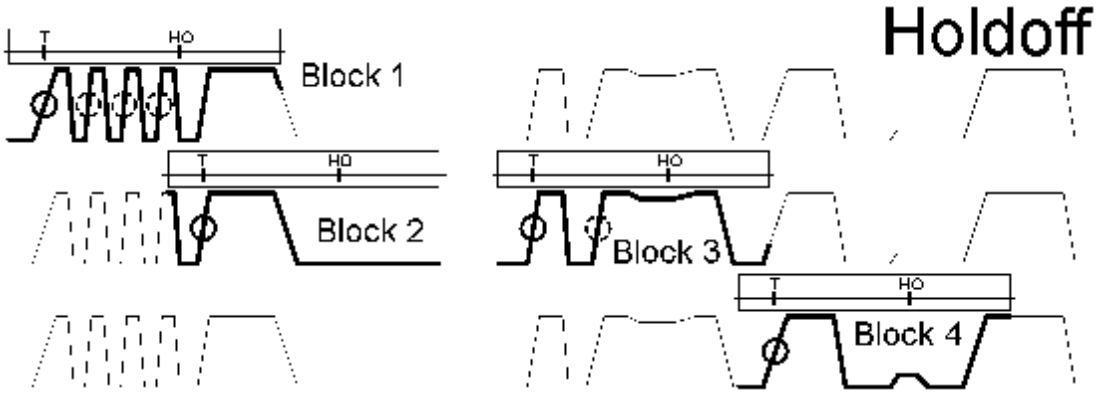
8.5.1 Normal

Falls die Option ***Normal*** gewählt wird, können **keine weiteren Einstellungen** gemacht werden. Diese Einstellung sollte möglichst vermieden werden, da sie leicht zur Überlastung des Computers führen kann.
Mit der zusätzlichen Option *Retrigger* oder *Holdoff* kann die **Überlappung der Datenblöcke optimiert** werden.



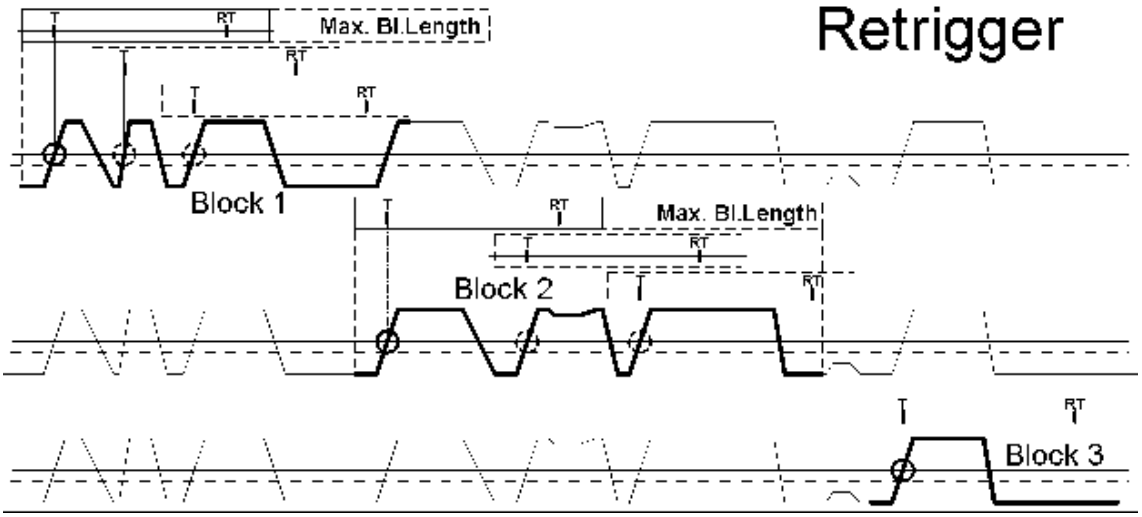
8.5.2 Holdoff

Mit der Holdoff Trigger-Option wird TranAX alle zusätzlichen **Trigger-Ereignisse bis zur *Holdoff*-Marke ignorieren**.

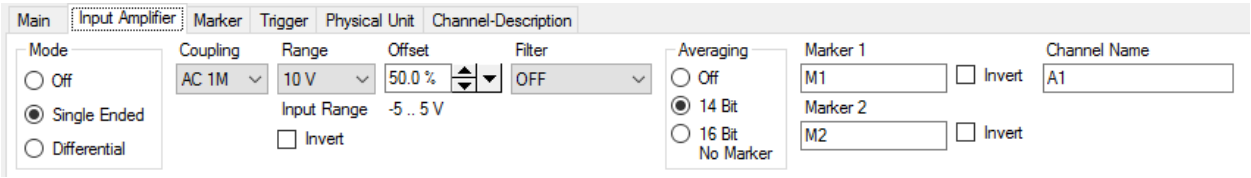


8.5.3 Retrigger

Bei Retrigger (wie auch bei Holdoff) kann die Marke entweder in **Sample-Längen** oder als **Zeitabstand** eingestellt werden. In diesem Zeitabschnitt werden **Trigger-Ereignisse ignoriert und nicht aufgezeichnet**. Falls in dieser Zeit aber ein neues Ereignis eintritt, wird die Retrigger-Marke relativ zu diesem Ereignis **neu gesetzt**. D.h. es erfolgt ein Retrigger und die Blocklänge wird vergrössert. Erst nachdem die Marke ohne Trigger-Ereignis überschritten wurde, kann ein neuer Block begonnen werden. Es kann eine maximale Posttrigger-Blocklänge eingestellt werden. TranAX triggert im Retrigger-Mode wie folgt:



9 Eingangsverstärker



Auf dieser Registerkarte stellt man folgende Kanalparameter ein:

- **Verstärker Mode**: Single-Ended (SE, Massebezogen), differentiell oder Ausgeschaltet
- **Kopplung**: DC, AC oder ICP/IEPE (Integrated Current Power für Piezo-Sensoren) Bei den 120MS und 240MS Modulen (TPCX und TPCE) kann zusätzlich die Eingangsimpedanz auf 50Ω gestellt werden. Bei allen anderen Modulen beträgt diese 1MΩ.
- **Eingangsspannungsbereiche**: Gesamter Bereich und Offset
- Eingangs-Tiefpass-**Filter** (inkl. Anti-Aliasing-Filter-Frequenz, als Option erhältlich)
- **Invertieren** des Eingangssignales
- **Kanal-Name**
- **Averaging**: Aus, 14Bit oder 16Bit
- **Namen** von **Marker**-Signalen (optionale Digitaleingänge)
- **Invertieren** der **Marker**-Signale

Diese Werte können für **jeden Kanal individuell** eingestellt werden. Beim Programmstart erkennt die Software automatisch, welche Kanäle im System installiert sind. Diese werden in der Tabelle aufgelistet.

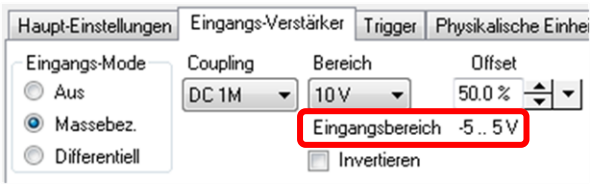
9.1 Mittelung (Averaging)

Der ADC läuft immer mit maximaler Abtastrate. Falls die gewählte Abtastrate unterhalb der maximalen Rate ist, werden alle Samples reduziert und gemittelt. **Das Signalrauschen wird entsprechend verkleinert**.

	Der Parameter **Averaging** wirkt immer gemeinsam auf **alle Kanäle** innerhalb eines Moduls.
	Für Anwendungen, welche eine Mittelung nicht erlauben (z.B. **Unterabtastung**), muss die **Mittelung abgeschaltet** werden.

9.2 Verstärker-Einstellungen

Der **Eingangsspannungsbereich** ergibt sich aus den zwei Werten **Bereich** und **Offset**. Der Bereich gibt den maximal erfassbaren Signalbereich an. Der Offset dient dazu, den Nullpunkt des Bereichs zu verschieben. Die resultierende minimal und maximal erfassbare Eingangsspannung wird sowohl unter den beiden Eingabefeldern als auch in der Tabelle als Information angezeigt. Der **Eingangsverstärker** von jedem Kanal kann **invertierend** arbeiten, d.h. die Polarität der Eingangsspannung wird umgekehrt. Um das entsprechende Signal zu bezeichnen, kann jedem Kanal ein **individueller Name** zugewiesen werden.



9.3 Marker (Digital-Eingänge)

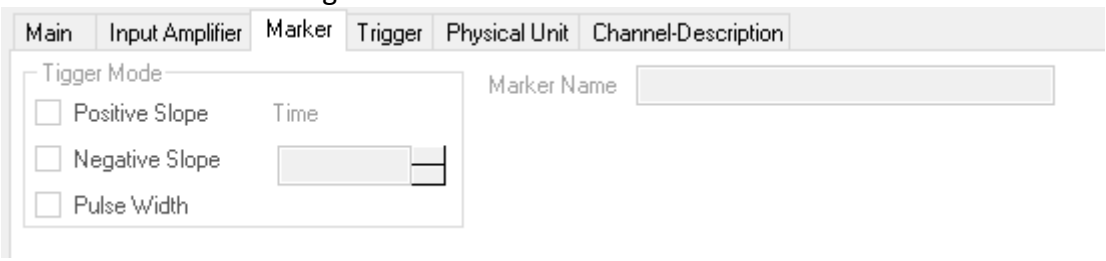
Marker-Eingänge können optional installiert werden.

Jeder Datenerfassungs-Kanal kann zwei Digital-Eingänge (Marker) haben. Das sind Eingänge, die ein 0/1-Signal aufweisen. Diese Signale können danach in einem **Marker Fenster** dargestellt werden.
In den Eingabefeldern Marker 1 und Marker 2 können die Namen der digitalen Eingänge definiert werden. Falls "Invert" aktiviert ist, wird das Signal vor der Erfassung invertiert. Dies ist insbesondere dann von Nutzen, wenn das Signal vorher von z.B. einem Optokoppler invertiert wurde. Siehe auch Kapitel Einschränkungen / Digitale Eingänge (Marker).

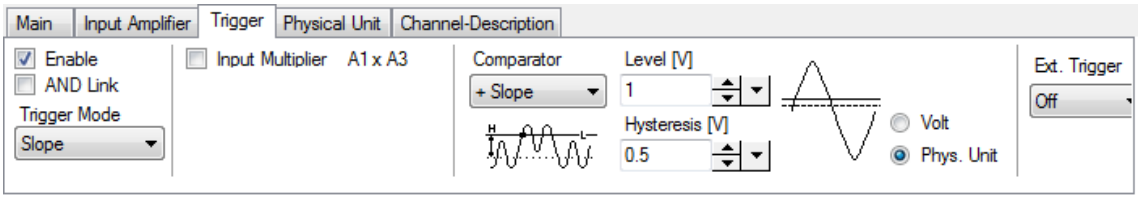
ℹ	**Invertierte Marker** werden mit einem **"\"** (Backslash, Rückwärtsschrägstrich) am Ende gekennzeichnet. "M1\" ist der invertierte Marker 1.
ℹ	Die Marker-Einstellungen können vorgenommen, werden auch wenn keine Marker-Option installiert ist.
⚠	Die Marker-Eingänge werden unterdrückt falls **Averaging auf 16-Bit** gesetzt ist.
⚠	Der entsprechende analoge Kanal muss im **Control Panel eingeschalten** sein (Karteireiter "Eingangs-Verstärker"), damit die Marker aufgezeichnet werden.

10 Marker

Die Funktionen in der Registerkarte "Marker" werden zurzeit noch nicht unterstützt.



11 Trigger

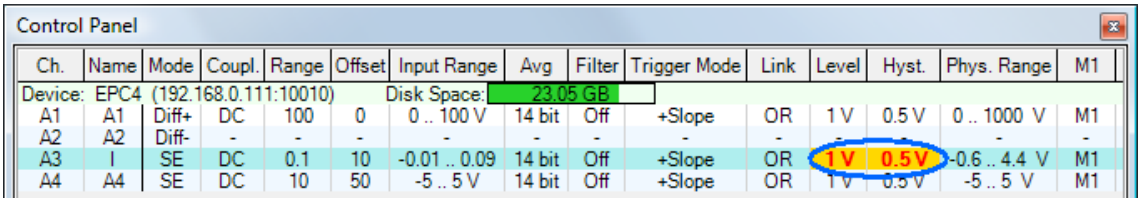


Die folgenden Parameter werden in dieser Registerkarte eingestellt:

- **Trigger Aktivieren: Der angewählte Kanal** ist nur eine aktive Triggerquelle, falls Aktivieren eingeschalten ist.
- AND Link: Die UND-Trigger-Verknüpfung wirkt nur innerhalb eines 4- oder 8-Kanal-Moduls.
- Trigger-Modi: Siehe das Kapitel Trigger-Modi für eine detaillierte Beschreibung der Trigger-Arten
- Multiplikation
- **Comparator** (Flanke)
- **Trigger-Pegel** [Volt oder Physikalische Einheit]
- **Trigger Hysteresis**

Die Trigger-Einstellungen gehören zu den sehr kritischen Parametern. Mangelhafte Einstellungen können zu ungewollten Aufnahmen führen oder die eingestellten Bedingungen sind nicht geeignet, um einen Trigger auszulösen.

Überprüfen Sie immer zuerst genau die Trigger-Einstellungen, falls ungewollte Aufnahmen gemacht werden oder die Triggerbedingung nicht erfüllt ist. Es empfiehlt sich, alle **ungewollten Triggerquellen zu deaktivieren**.
Die Trigger-Parameter können für jede Triggerquelle separat eingestellt werden. Bei Programmstart erkennt die Software automatisch, welche triggerfähigen Kanäle im System installiert sind. Die triggerfähigen Kanäle haben in der Tabelle Einträge unter Trigger-Modus und -Pegel.
Je nach gewähltem Trigger Modus können entweder Pegel und Hysterese (+Flanke, -Flanke, ±Flanke) oder ein unterer und ein oberer Pegel (Fenster Eintritt, Fenster Austritt) in Volt oder in der physikalischen Einheit eingegeben werden. Mit der Hysterese-Einstellung kann verhindert werden, dass bei verrauschten Signalen auf die falsche Flanke getriggert wird. Die aktuellen Einstellungen werden zur Information in den Grafik-Symbolen dargestellt.

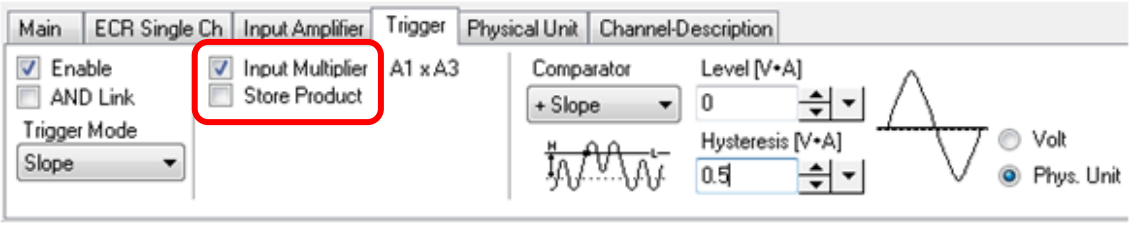


Control Panel

Ch.	Name	Mode	Coupl.	Range	Offset	Input Range	Avg	Filter	Trigger Mode	Link	Level	Hyst.	Phys. Range	M1
Device:	EPC4	(192.168.0.111:10010)				Disk Space:	23,05 GB							
A1	A1	Diff+	DC	100	0	0 .. 100 V	14 bit	Off	+Slope	OR	1 V	0.5 V	0 .. 1000 V	M1
A2	A2	Diff-	-	-	-	-	-	-	-	-	-	-	-	-
A3	I	SE	DC	0.1	10	-0.01 .. 0.09	14 bit	Off	+Slope	OR	1 V	0.5 V	-0.6 .. 4.4 V	M1
A4	A4	SE	DC	10	50	-5 .. 5 V	14 bit	Off	+Slope	OR	1 V	0.5 V	-5 .. 5 V	M1

Falls die Spalten Pegel und Hysterese gelb/rot erscheinen, heisst das, dass die Einstellungen ungültig sind, z.B. ist der Eingangsbereich bis 0.5 V und der Pegel wurde bei 1 V eingestellt, d.h. dieser Pegel kann gar nie erreicht werden.

11.1 Eingangs-Multiplizierer



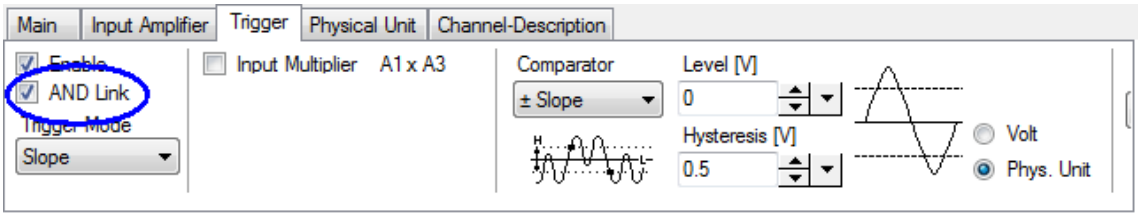
Die Aktivierung des Parameters **Multiplikation** bewirkt, dass die Hardware laufend die **digitalisierten Signale von zwei Kanälen** (z.B. A1 und A3 oder A2 und A4) miteinander multipliziert. Das daraus resultierende Produkt-Signal wird dann anstelle des Original-Signals von Kanal A1 bzw. A2 zum Trigger-Diskriminator geführt. Der Bereich und die Auflösung des Produkts hängt von den Einstellungen "Eingangs-Bereich", "Offset", "physikalischer Skalierungsfaktor" und "physikalische Einheit" der beiden zugehörigen Kanäle ab.
Der volle Bereich und damit die Auflösung kann nur maximal bis zur Hälfte ausgeschöpft werden. Dies ist der Fall, wenn bei beiden Kanälen der Offset auf 0% oder auf 100% gestellt ist. Je nachdem ergibt das dann eine Produkt-Kurve von 0 bis +Max oder bis -Max. Eine Produkt-Kurve von -Max bis +Max kann es nie geben.
Man erreicht nur ein Viertel der maximalen möglichen Auflösung, wenn bei beiden Kanälen der Offset auf 50% (=Bereichsmitte) gestellt ist.

Es gilt zu beachten, dass der Hardware-Multiplizierer die Konstante bei der physikalischen Skalierung nicht berücksichtigen kann. Die Konstante sollte deshalb bei beiden Kanälen auf 0 gestellt werden.

Durch Einschalten eines weiteren Parameters **Produkt speichern** wird beim ersten der beiden Kanäle (z.B. A1) die berechnete Signalkurve anstelle des Original-Signals aufgezeichnet. Beim zweiten zugehörigen Kanal (z.B. A3) erfolgt die Aufzeichnung unverändert.

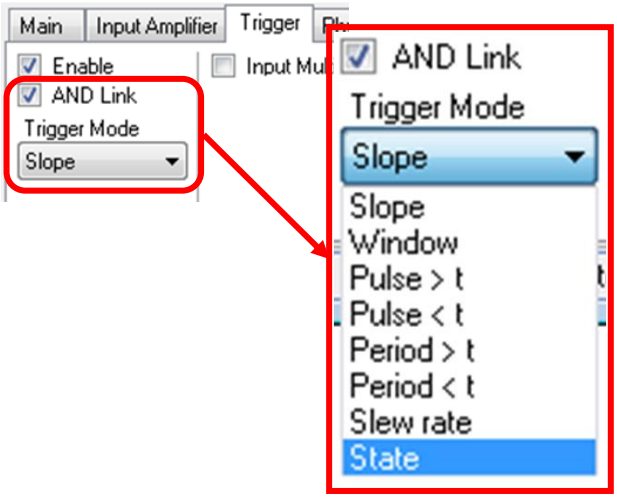
Die Amplituden-Auflösung der Produkt-Kurve beträgt auch 14-Bit bzw. 16-Bit. Falls die Original-Signale nicht den vollen Aussteuerungsbereich der Verstärker ausnützen, kann sich die Auflösung der resultierenden Kurve nicht akzeptierbar stark verringern. Es empfiehlt sich also für beide Signale jeweils den **Messbereich** so zu wählen, dass die **Verstärker knapp nicht übersteuert** werden.

11.2 AND Link (Logische UND Verknüpfung)



Die UND-Verknüpfung steht nur zur Verfügung, wenn die Option Advanced Trigger installiert ist.

Standardmässig sind alle aktiven Triggerquellen in ODER-Logik verknüpft. Das heisst, dass **jede aktive Triggerquelle** innerhalb eines Systems **Trigger auslösen kann**.
Falls für spezielle Triggerbedingungen die UND-Logik erwünscht ist, so können alle Kanäle eines Moduls, wie zum Beispiel A1 - A8, in diesem Fall die Eingangskanäle eines 8-Kanal TPCX/TPCE-Modules, in einer UND-Kombination konfiguriert werden. Die UND-Verknüpfung lässt sich beliebig kombinieren mit all den anderen zur Verfügung stehenden Trigger Modi.



Der Trigger Modus **Zustand** steht zur Auswahl, sobald der **AND Link aktiviert** ist.

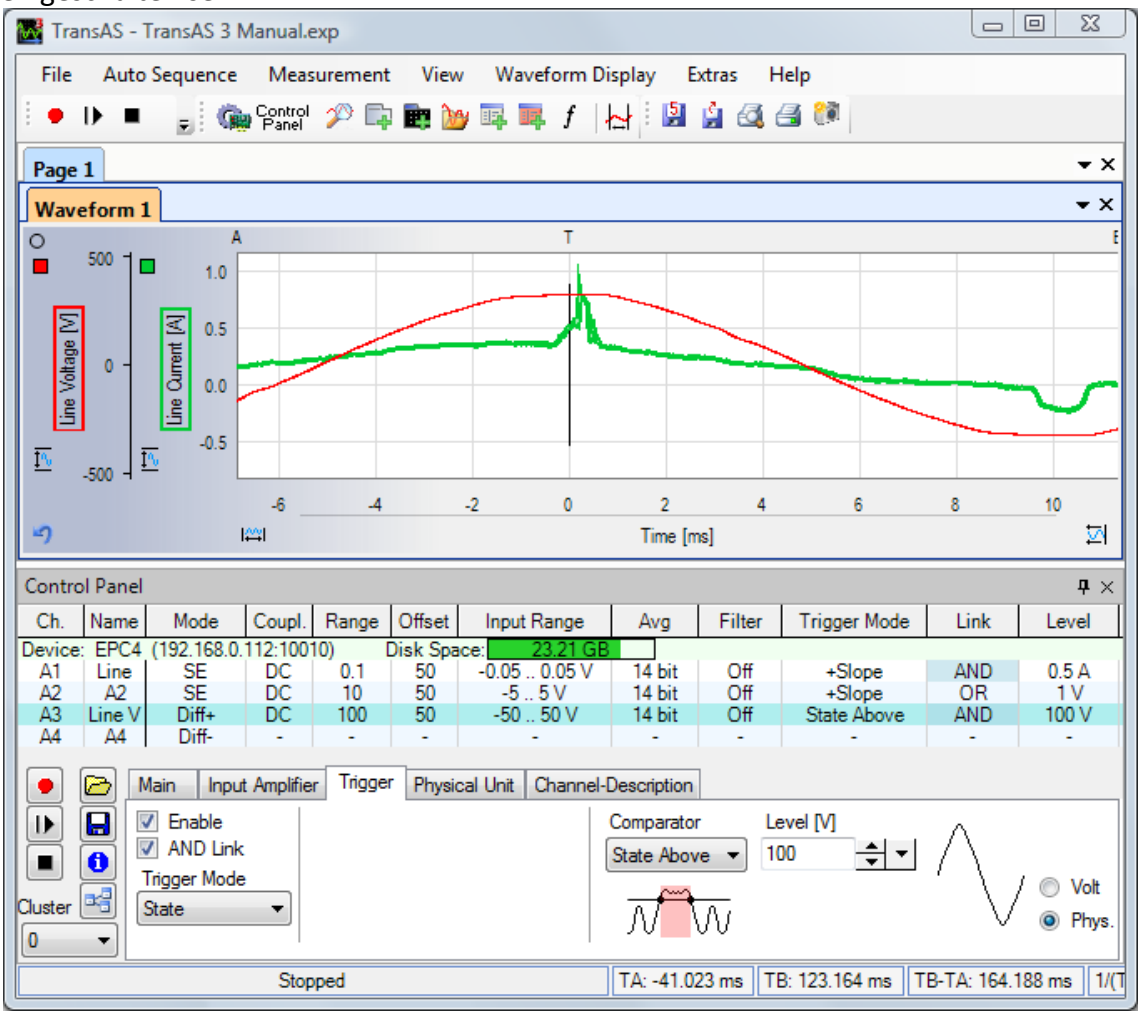
Der **Zustands-Trigger** kann als **Qualifizierer** für weitere Triggerquellen eingesetzt werden. Nur wenn der Zustand dieses Kanales zutrifft wird auf den **andern** (UND-Verknüpften) **Trigger-Quellen getriggert** sobald eine von deren Bedingungen erfüllt ist.

Falls nur bei **einem Kanal "AND Link"** aktiviert wurde, so verhält sich diese Trigger-Quelle im System genau gleich wie bei einer ODER-Verknüpfung.
Es müssen **mindestens zwei Kanäle** "AND Link" zugeschaltet haben, um diese Funktion sinnvoll zu verwenden.

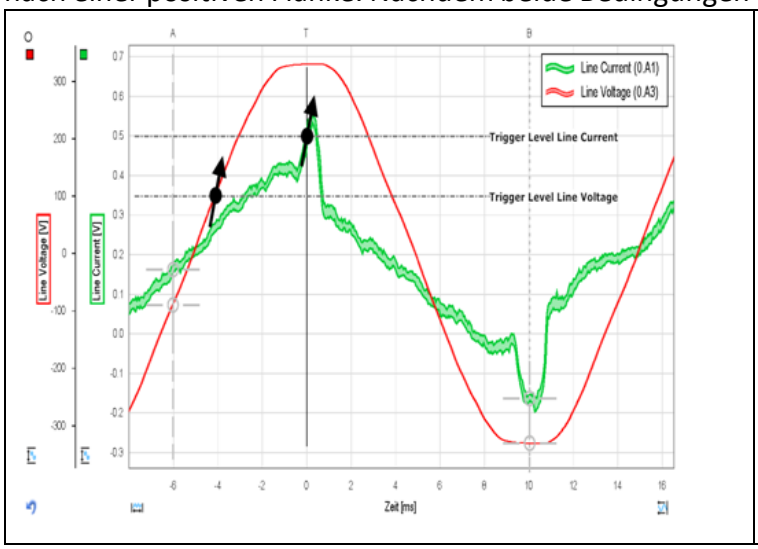
Die Standard ODER-Verknüpfung funktioniert mit allen aktiven Trigger-Kanälen, auch bei mehreren Transientenrekordern, die via Sync-Link Box synchronisiert sind. Die UND-Verknüpfung hingegen lässt sich ausschliesslich pro Kanalgruppe aufsetzen, also pro 4- oder 8-Kanal Modul.

11.2.1 Beispiel 1: AND-Link (Zustand mit Flanke)

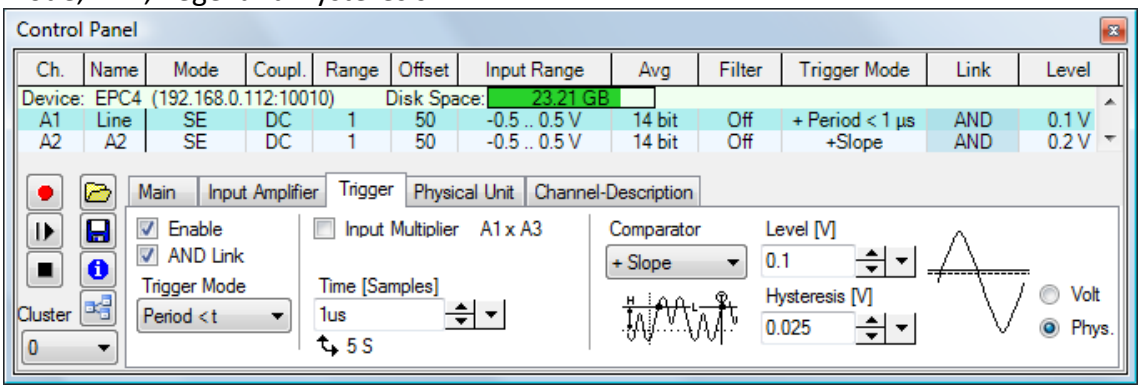
Nur wenn das **Signal "*Line Voltage*"** (rote Kurve) **oberhalb** von **100 V** liegt UND dann die positive Flanke des **Kanales "*Line Current*" 0.5 A** durchfährt wird ein **Trigger ausgelöst**.
Dazu müssen beide Kanäle von derselben Gruppe sein und bei beiden Kanälen muss AND Link eingeschalten sein.



In diesem Beispiel sucht Kanal 1 (grüne Kurve) nach Perioden kleiner als 1 µs und Kanal zwei sucht nach einer positiven Flanke. Nachdem beide Bedingungen erfüllt sind, wird der Trigger ausgelöst.

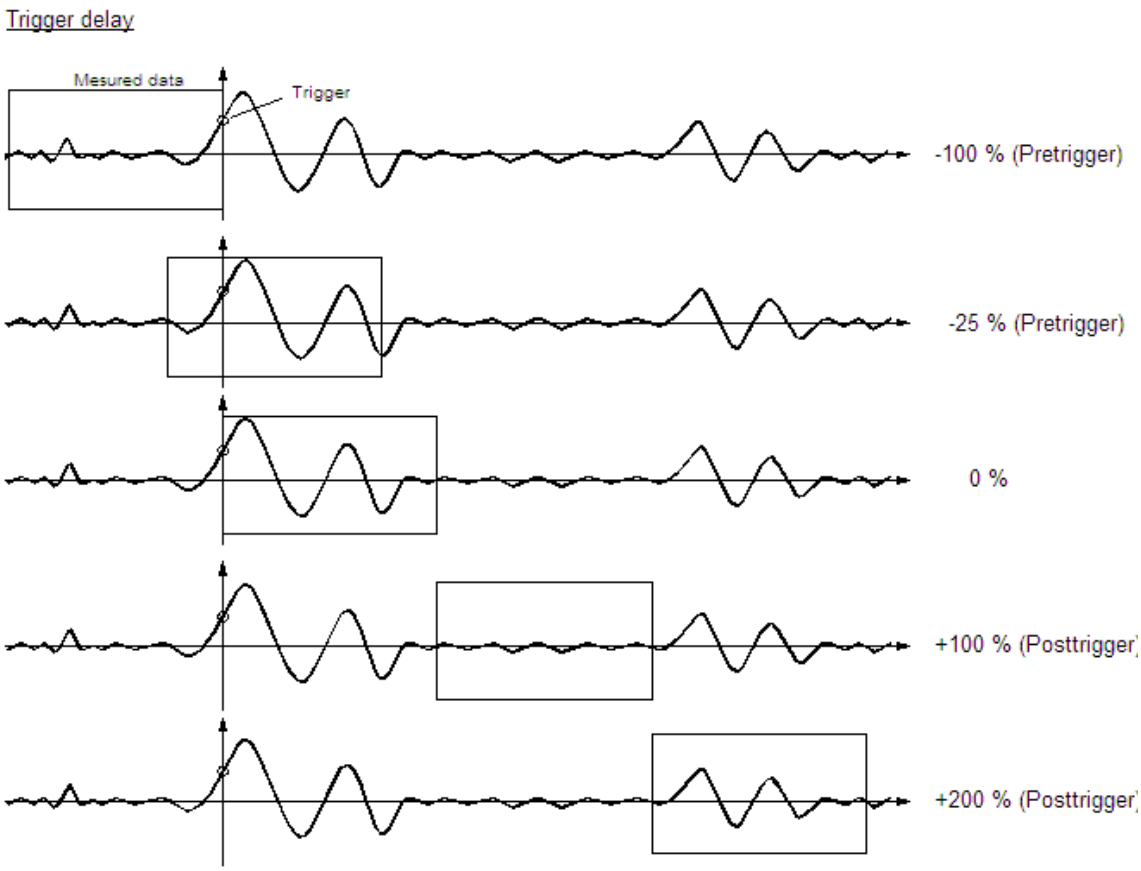
	A1:+Periode<t, t<1µs A2: Flanke, +Flanke

Das Control Panel wiederspiegelt die momentanen Trigger-Einstellungen in den Kolonnen Trigger Mode, Link, Pegel und Hysteresis.



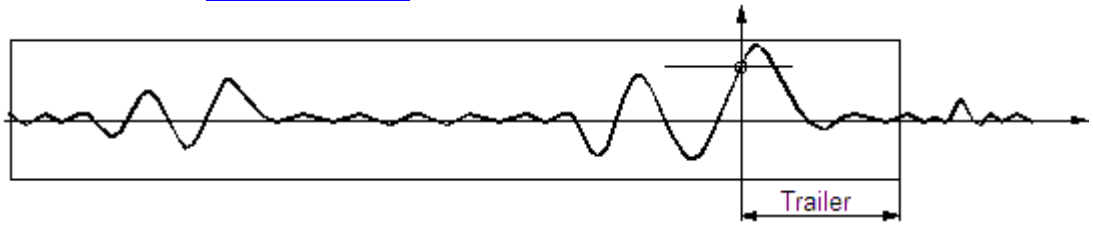
11.3 Pre- und Post-Trigger (Triggerverzögerung)

Diese Datenaufnahme-Parameter werden bei Scope-, Multi-Block- und ECR-Mode verwendet. Die Position des **Aufnahmefensters** (oder Blocks) relativ zum **Triggerzeitpunkt** kann in gewissen Grenzen eingestellt werden. Dabei spricht man von Pre-Trigger, wenn die Aufnahme Teile des Signals vor dem Triggerzeitpunkt enthält. Werden nur Signalanteile nach dem Triggerzeitpunkt aufgezeichnet, spricht man von Post-Trigger. Diese Triggerverzögerung wird in Prozenten der Gesamtaufnahmedauer angegeben. Die TPCX/TPCE-Hardware erlaubt Triggerverzögerungen von -100% bis +200%.



Im kontinuierlichen Mode gibt es keinen Pre- oder Post-Trigger. An dessen Stelle tritt der Stopp-Trigger und der **Nachlauf** (Trailer). Im kontinuierlichen Mode kann die Aufnahme beendet werden, wenn die TPCX/TPCE-Hardware ein Trigger-Ereignis erkennt. Dann spricht man von einem Stopp-Trigger. Manchmal ist es wünschenswert, die Aufnahme dann noch eine Weile fortzusetzen, bevor sie definitiv beendet wird. Dazu dient der Nachlauf-Parameter. Dieser gibt an, wie

viele Samples nach dem Stopp-Trigger zusätzlich noch aufgezeichnet werden. Die Datenerfassungs-Hardware erlaubt TranAX-Einstellungen von 0 (kein Nachlauf) bis 16 MSamples. Diese Option ist auch im ECR Dual-Mode enthalten.

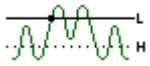


11.4 Trigger-Modi

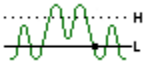
Folgende Trigger-Arten können jeweils für alle triggerfähigen Kanäle individuell eingestellt werden. **"L"** entspricht dem **Trigger-Level** und **H** dem einstellbaren **Hystereseband**.

11.4.1 Flanke

Mit den **Flanken-Einstellungen** kann man entweder die positive, negative oder beide Flanken des Trigger-Signals auswählen. TranAX triggert nur wenn das Signal das **Hystereseband durchlaufen** hat.



+ Flanke

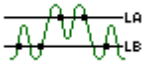


- Flanke

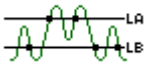
± Flanke

11.4.2 Fenster

Mit der **Fenster-Eintritt** Trigger-Wahl löst TranAX jedes Mal den Trigger aus, falls das Signal in das Fenster eintritt. Entsprechendes gilt für **Fenster-Austritt**.



Fenster-Eintritt

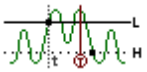


Fenster-Austritt

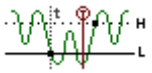
Die folgenden Trigger-Arten benötigen die Option Advanced-Trigger.

11.4.3 Puls > Zeit

Sobald ein positiver oder negativer Puls erkannt wurde, wird ein Trigger-Ereignis nur ausgelöst, wenn der Puls grösser ist als die angegebene Zeit.
Bitte beachten: Um das Ende eines Pulses zu bestimmen, muss die eingestellte Hysterese beachtet werden!



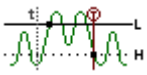
+ Puls > t



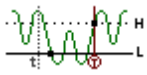
- Puls > t

11.4.4 Puls < Zeit

Es wird nur getriggert, falls die Pulsweite kleiner ist als die angegebene Zeit.



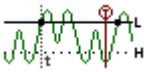
+ Puls < t



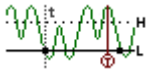
- Puls < t

11.4.5 Periode > Zeit

TranAX triggert nur, wenn eine Periode grösser ist als die eingestellte Zeit. Die Hysterese ermöglicht eine Unterdrückung ungültiger Perioden (z.B. hochfrequentes Rauschen).



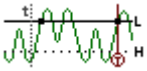
+ Periode > t



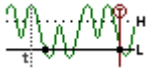
- Periode > t

11.4.6 Periode < Zeit

Wenn eine Periode kürzer als die eingestellte Zeit ist, wird ein Trigger-Ereignis ausgelöst.



+ Periode < t



- Periode < t

11.4.7 Flankensteilheit

Mit der Flankensteilheit-Option wird der Trigger bei Überschreiten einer gewissen Anstiegs- oder Abfallsteilheit ausgelöst. Sie wird hauptsächlich gebraucht, um schnelle Parasiten oder Spitzen auf langsamen Signalen zu erkennen.
Die Flankensteilheit kann mittels **Delta Samples** (Delta Zeit) und **Delta-Y** (Delta Amplitude) eingestellt werden. **Delta-Y** sollte immer doppelt so gross sein wie das erwartete Rauschen des Signals. Der **Delta Zeit Parameter ist auf 1024 Samples limitiert**. Die Flankensteilheit-Werte werden auch in der Kanalliste in Kolonne *Trigger Mode* angezeigt.

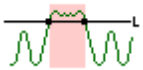
+ Flankensteilheit

- Flankensteilheit

± Flankensteilheit

11.4.8 Zustand

Der Trigger-Mode **Zustand** steht nur zur Auswahl, wenn **AND Link** angewählt ist. Er wird in der Regel in einer UND-Verknüpfung als Qualifizierer verwendet, um die Trigger-Auslösung durch andere Quellen von einem Signal-Zustand abhängig zu machen. Die UND-Verknüpfung kann nur innerhalb eines TPCX/TPCE -Modules angewendet werden.


Grösser als


Kleiner als

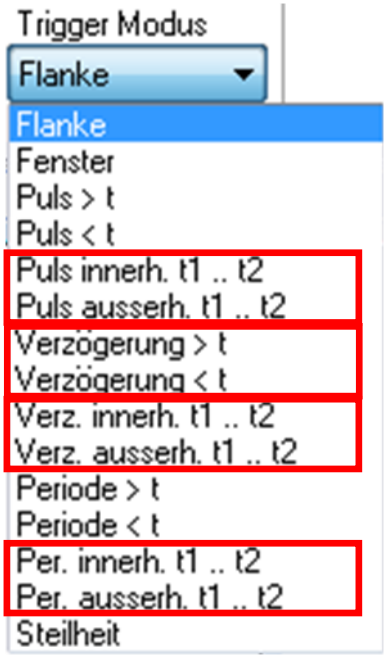
Ausserdem steht bei einem TPCX/TPCE-Instrument ein **externer Trigger-Eingang** zur Verfügung (TTL). Mit **einem externen Disarm-Eingang** kann die Triggerauslösung durch ein digitales Signal (TTL) verhindert und wieder freigegeben werden. Die diesbezügliche Steckerbelegung findet man im Hardware-Manual.

11.5 Erweiterte Trigger-Funktionen (Übersicht)

Zusätzlich zu den bestehenden sieben Trigger-Modi wurden **acht weitere** implementiert. Diese benötigen ebenfalls die Option *Advanced Trigger*. Auf die bereits vorhandenen Trigger-Modi wird nur am Rande eingegangen.

Die acht neuen Trigger Modi:

- Puls innerhalb t1 .. t2
- Puls ausserhalb t1 .. t2
- Verzögerung > t
- Verzögerung < t
- Verzögerung innerhalb t1 .. t2
- Verzögerung ausserhalb t1 .. t2
- Periode innerhalb t1 .. t2
- Periode ausserhalb t1 .. t2



Im Control-Panel können die Zeiten (t, t1, t2) auch als **Anzahl Samplings** eingegeben werden. Das Programm errechnet damit (multipliziert mit der Abtastrate) die entsprechenden Zeiten. **Intern** werden diese als **Anzahl Samplings** und nicht als Zeiten verwaltet.

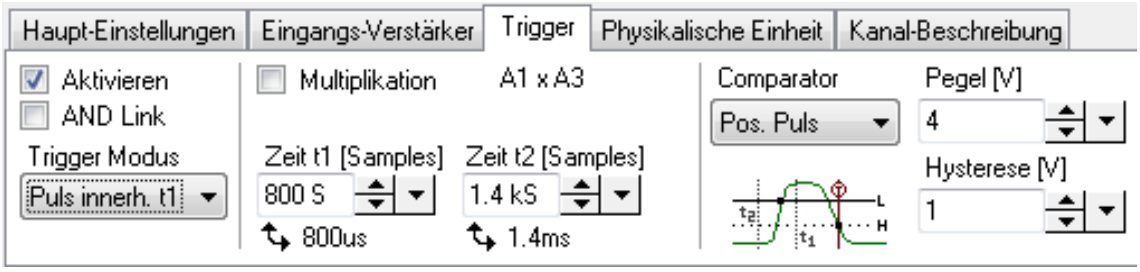
Die eingestellten Zeiten ändern sich, wenn die Zeitbasis verstellt wird!

Um die erweiterten Trigger-Modi zu nutzen, muss die Installierte Software eventuell aktualisiert werden. Die folgenden Versionen werden als **Minimum** vorausgesetzt:

TranAX:	**3.2.1.624**	(Menü ***"Hilfe" / "Info"***)
TPC-Server:	**1.3.2**	(Control Panel /)

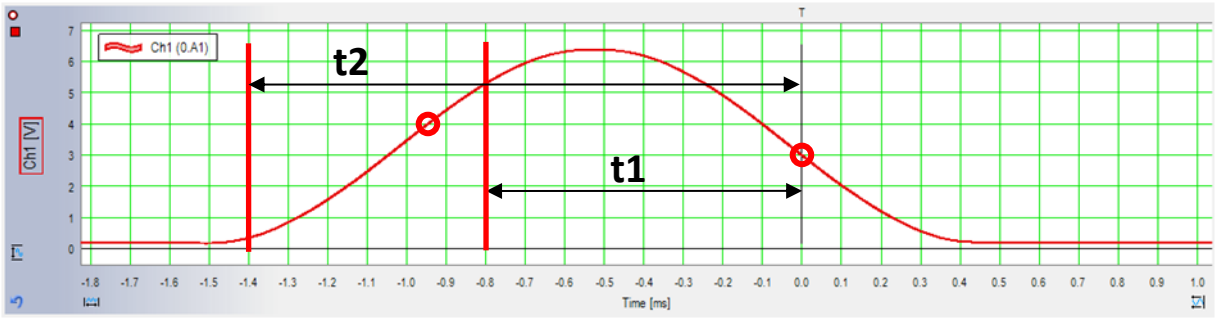
11.5.1 Puls innerhalb t1 .. t2

Der Trigger löst aus, wenn die **Pulsbreite innerhalb** der eingestellten Zeiten **t1 und t2** liegt. In diesem Beispiel sind die Zeiten t1 = 0.8ms und t2 = 1.4ms eingestellt. Abtastrate ist auf 1MS/s eingestellt. 800 Samples entsprechen somit genau 0.8ms, respektive 1.4kS entsprechen den 1.4ms.



Einstellungen für Kanal 1.

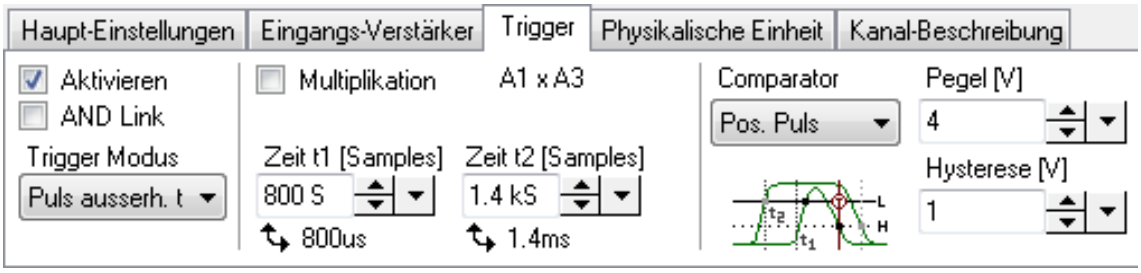
Da auf eine positive Flanke getriggert wird (Pos. Puls), ist die Bedingung beim sinkenden Signal erfüllt, der Trigger-Punkt wird auf die sinkende Flanke gesetzt, und zwar auf den Wert **Pegel minus Hysterese**. In diesem Beispiel bei 3V.



Die Pulsbreite beträgt ca.0.95ms und ist somit innerhalb der beiden Zeiten t1 und t2 von 0.8ms und 1.4ms.

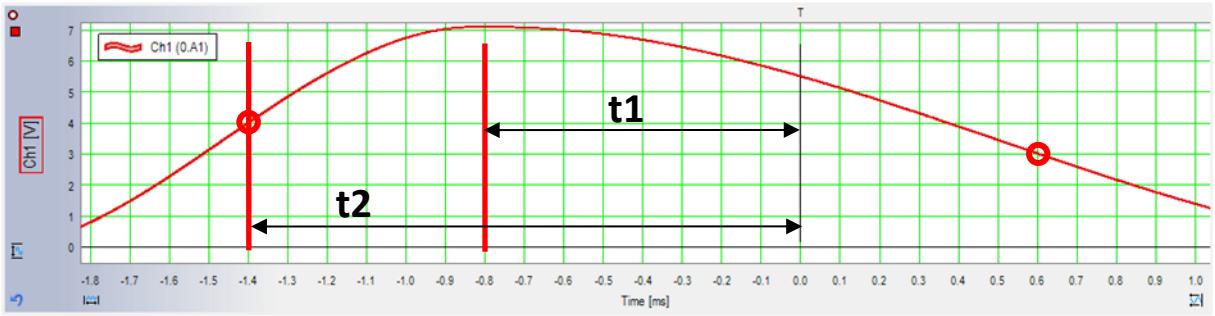
11.5.2 Puls ausserhalb t1 .. t2

Der Trigger löst aus, wenn sich die Pulsbreite **ausserhalb** der eingestellten Zeiten t1 und t2 befindet. D.h. die Pulsbreite muss kleiner t1 oder grösser t2 sein, damit die Trigger-Bedingung erfüllt ist (Pulsbreite <t1 oder Pulsbreite > t2).



Einstellungen für Kanal 1.

*Hier ist die **Pulsbreite < t1**.*



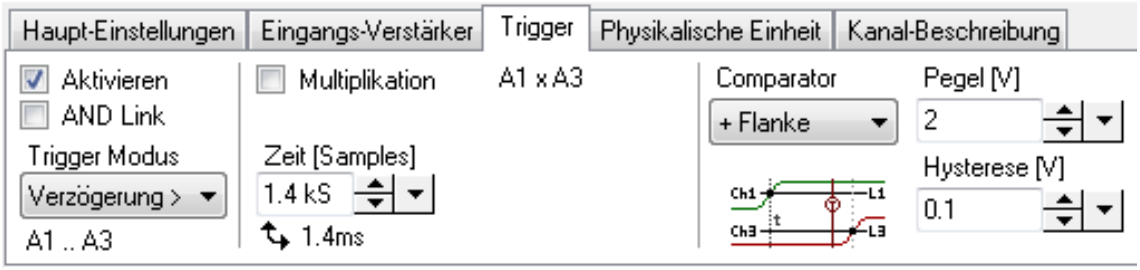
*In diesem Bild ist die **Pulsbreite > t2**.*

Der Trigger-Zeitpunkt befindet sich genau **t2** hinter dem Kriterium für den Beginn des Pulses (hier pos. Flanke, Pegel = 4V). Das Signalverhalten zeigt also genau **beim Trigger-Nullpunkt keinen Trigger-Hinweis**. Das Ende des Pulses (hier neg. Flanke bei 3V) befindet sich dahinter, ev. sogar im Unendlichen oder wird je nach Block-Länge nicht mehr aufgezeichnet.

11.5.3 Verzögerung > t

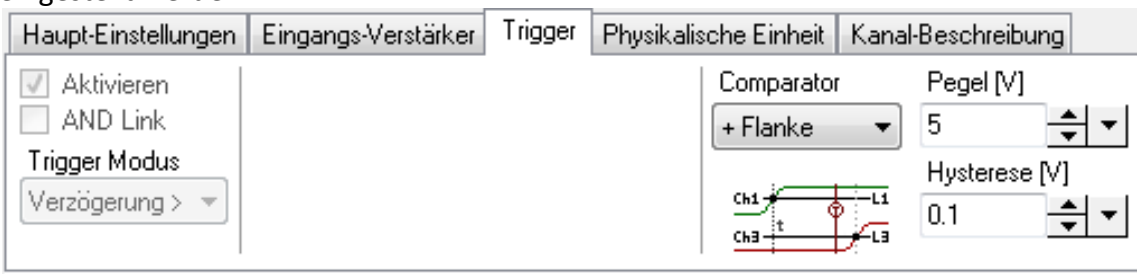
Dieser Trigger-Mode verwendet die Signale der Kanäle 1 und 3 (bzw. 2 und 4). Es wird dabei erfasst, ob die **Zeit** zwischen dem Erfüllen der Trigger-Bedingungen von **Kanal 1** und **Kanal 3 grösser** als die eingestellte **Zeit t** ist. Ist dies der Fall, wird ein Trigger ausgelöst.
Zusätzlich zu der **Zeit t** müssen für die beiden Kanäle 1 und 3 die Einstellungen des Komparators vorgenommen werden (Flanke, Pegel und Hysterese).

Kanal 1 und 3 sowie Kanal 2 und 4 werden für diesen Trigger-Modus kombiniert. Andere Kanal-Kombinationen sind nicht möglich.

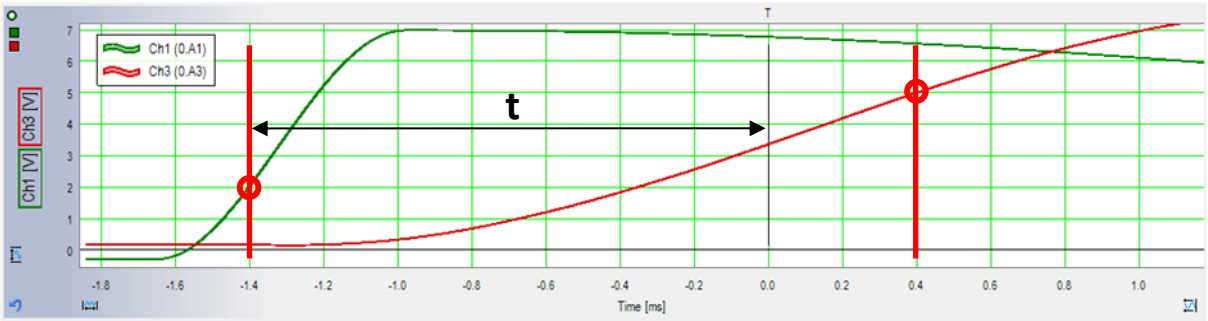


Einstellungen für Kanal 1.

Für den zugehörigen Kanal 3 (bzw. 4) kann keine Änderung am Trigger-Modus vorgenommen werden, dieser wird von Kanal 1 (bzw. 2) bestimmt. Beim zugehörigen Kanal (3 bzw. 4) kann die Flanke unabhängig "+", "-" oder "±" gewählt werden. Ebenso können der Pegel und die Hysterese eingestellt werden.



Einstellungen für Kanal 3. Der Trigger-Modus ist an jenen von Kanal 1 gekoppelt.

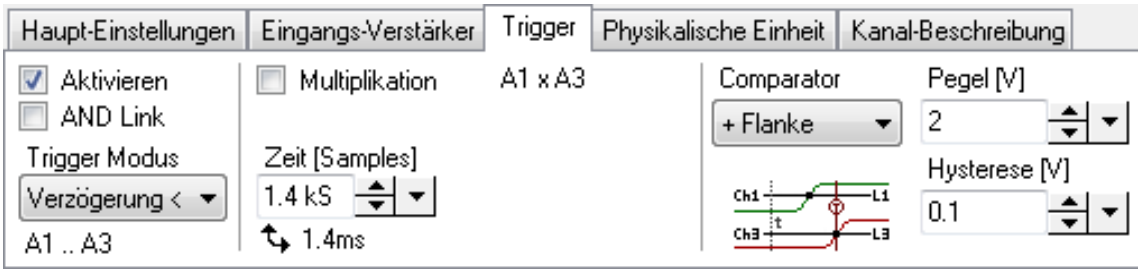


*In diesem Bild ist die **Verzögerung > t**.*

Der Trigger-Nullpunkt befindet sich genau um die **Zeit t** hinter dem Trigger-Kriterium für den ersten Kanal (hier pos. Flanke, 2V). Das Signal-Verhalten zeigt also genau beim Trigger-Nullpunkt keinen Trigger-Hinweis. Das Trigger-Kriterium vom zugehörigen Kanal (hier pos. Flanke, 5V) befindet sich dahinter, ev. sogar im Unendlichen oder wird je nach Block-Länge nicht mehr aufgezeichnet.

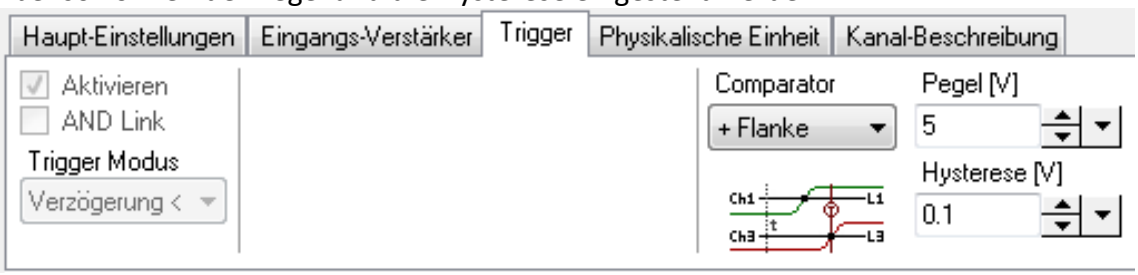
11.5.4 Verzögerung < t

Hier wird ein Trigger ausgelöst, wenn die **Verzögerung kleiner** der eingestellten **Zeit t** ist. Zusätzlich zu der Zeit t müssen für die beiden Kanäle 1 und 3 (bzw. 2 und 4) die Einstellungen des Komparators vorgenommen werden (Flanke, Pegel und Hysterese).

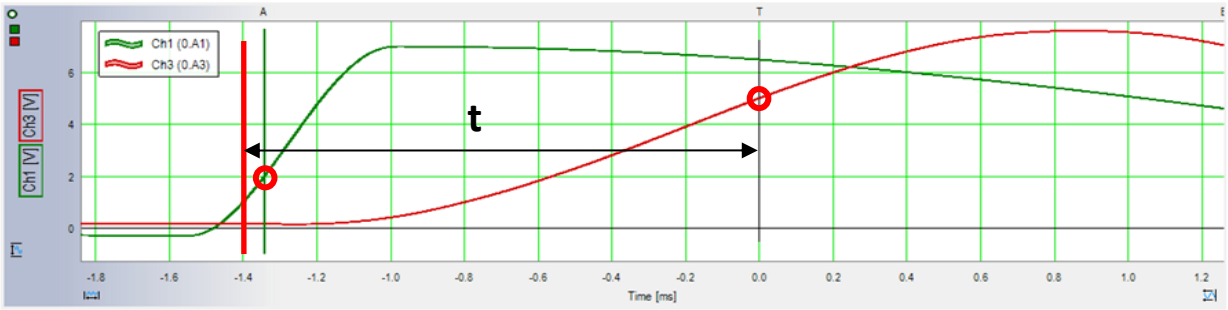


Einstellungen für Kanal 1.

Beim zugehörigen Kanal (3 bzw. 4) kann die Flanke unabhängig "+", "-" oder "±" gewählt werden. Ebenso können der Pegel und die Hysterese eingestellt werden.



Einstellungen für Kanal 3. Der Trigger-Modus ist an jenen von Kanal 1 gekoppelt.

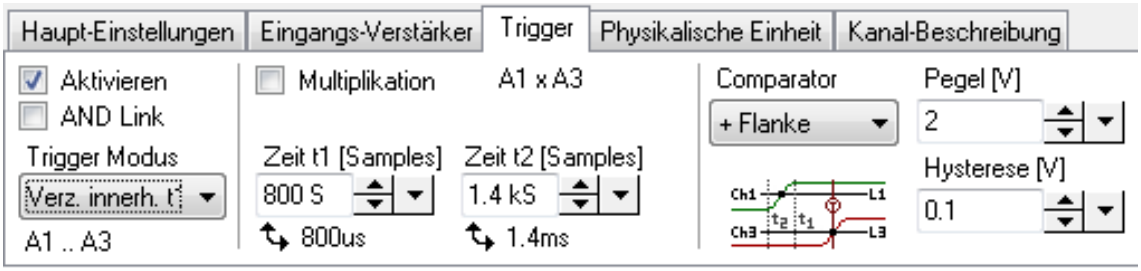


In diesem Bild ist die Verzögerung ***<t.***

11.5.5 Verzögerung innerhalb t1 .. t2

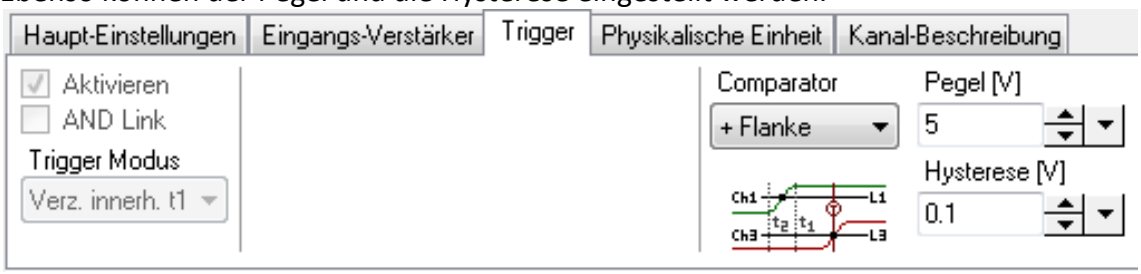
Dieser Trigger-Mode verwendet die Signale der Kanäle 1 und 3 (bzw. 2 und 4). Es wird dabei erfasst, ob die **Zeit** zwischen dem Erfüllen der Trigger-Bedingungen von **Kanal 1** und **Kanal 3 innerhalb** der eingestellten **Zeiten t1 und t2** ist. Ist dies der Fall, wird ein Trigger ausgelöst.

Zusätzlich zu den **Zeiten t1 und t2** müssen für die beiden Kanäle 1 und 3 (bzw. 2 und 4) die Einstellungen des Komparators vorgenommen werden (Flanke, Pegel und Hysterese).

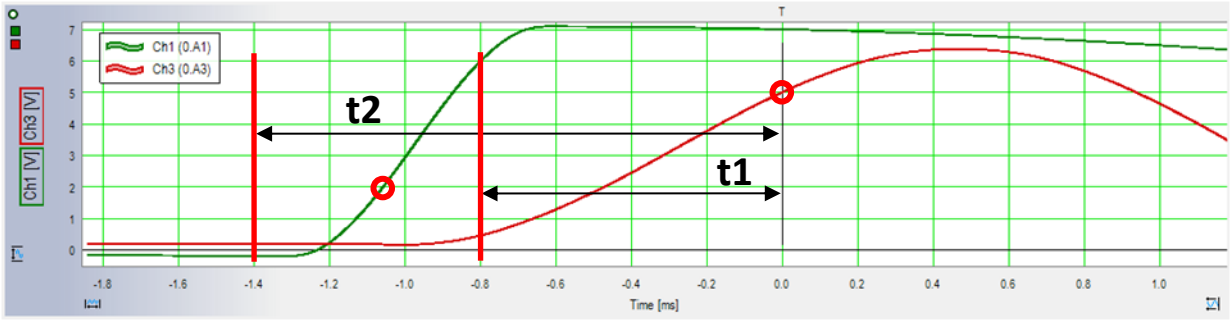


Einstellungen für Kanal 1.

Beim zugehörigen Kanal (3 bzw. 4) kann die Flanke unabhängig "+", "-" oder "±" gewählt werden. Ebenso können der Pegel und die Hysterese eingestellt werden.



Einstellungen für Kanal 3. Der Trigger-Modus ist an jenen von Kanal 1 gekoppelt.

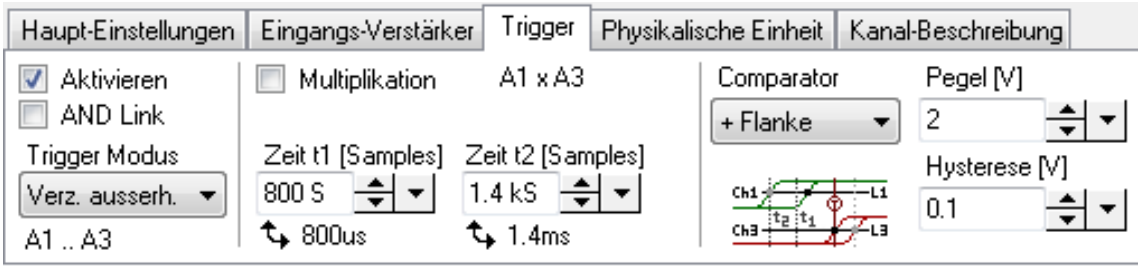


In diesem Bild ist die Verzögerung ca. 1.1ms (d.h. innerhalb von 0.8.und 1.4ms).

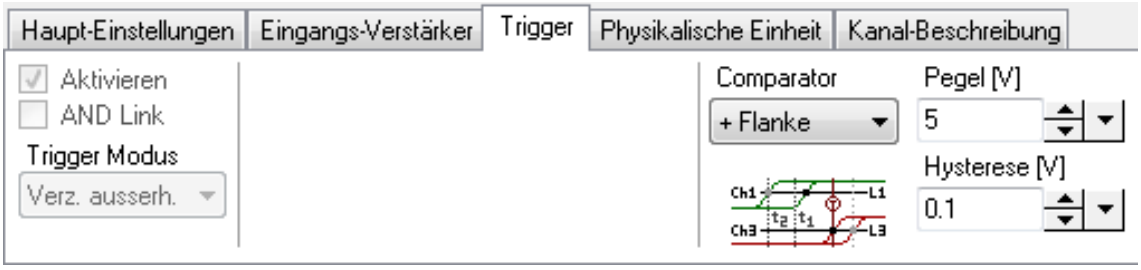
11.5.6 Verzögerung ausserhalb t1 .. t2

Dieser Trigger-Mode verwendet die Signale der Kanäle 1 und 3 (bzw. 2 und 4). Es wird dabei erfasst, ob die **Zeit** zwischen dem Erfüllen der Trigger-Bedingungen von **Kanal 1** und **Kanal 3 ausserhalb** der eingestellten **Zeiten t1 und t2** ist. Ist dies der Fall, wird ein Trigger ausgelöst.

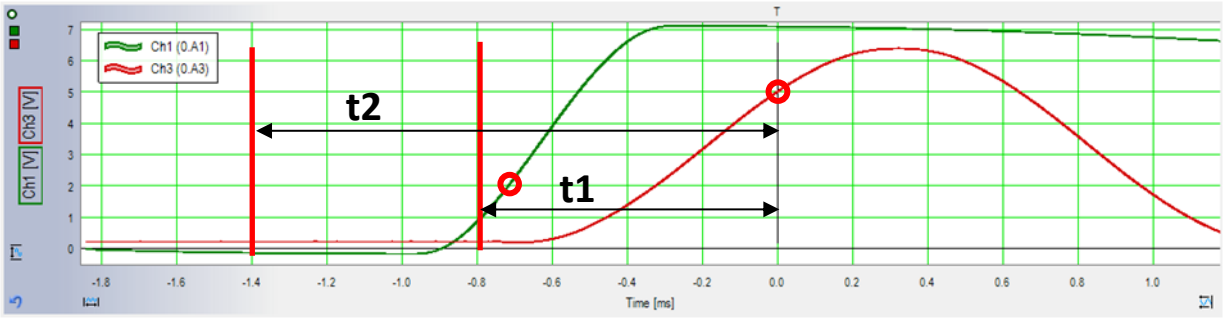
Zusätzlich zu den **Zeiten t1 und t2** müssen für die beiden Kanäle 1 und 3 (bzw. 2 und 4) die Einstellungen des Komparators vorgenommen werden (Flanke, Pegel und Hysterese).



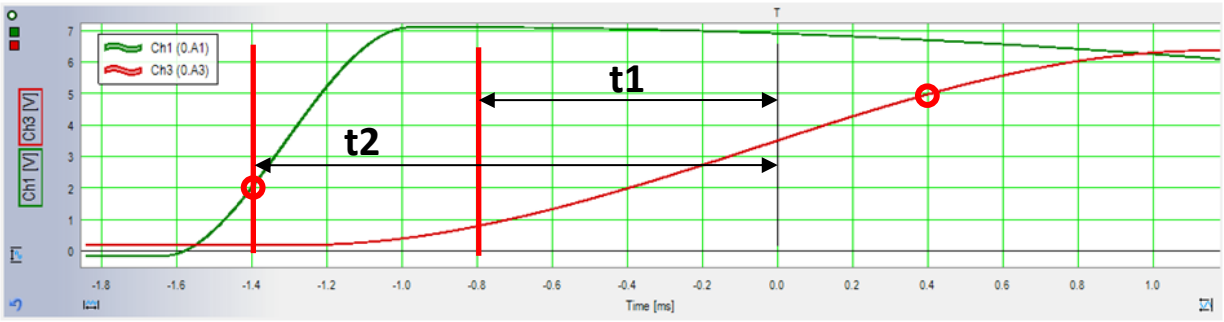
Einstellungen für Kanal 1.



Einstellungen für Kanal 3. Der Trigger-Modus ist an jenen von Kanal 1 gekoppelt.



*In diesem Bild ist die **Verzögerung < t1***

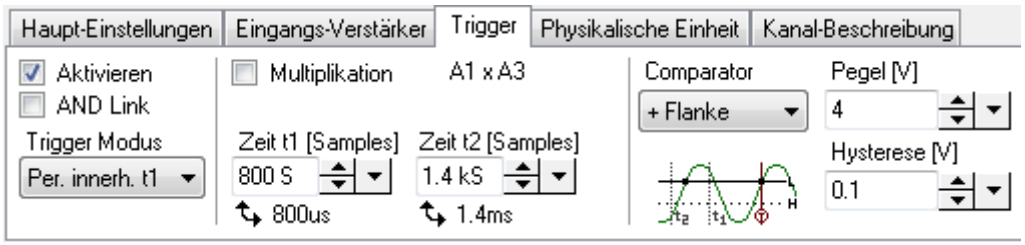


*In diesem Bild ist die **Verzögerung >t2**.*

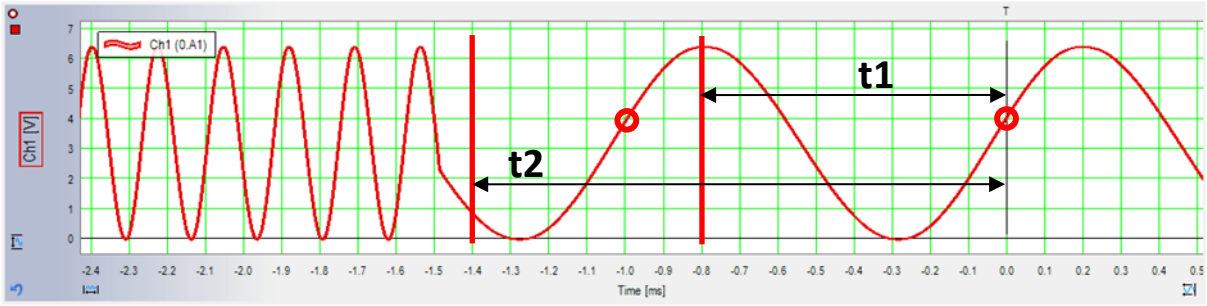
Der Trigger-Zeitpunkt befindet sich genau t2 hinter dem Trigger-Kriterium für den ersten Kanal (hier pos. Flanke, Pegel = 2V). Das Signal-Verhalten zeigt also genau beim Trigger-Nullpunkt keinen Trigger-Hinweis. Das Trigger-Kriterium für den zugehörigen Kanal (hier pos. Flanke, Pegel = 5V) befindet sich dahinter, ev. sogar im Unendlichen oder wird je nach Block-Länge nicht mehr aufgezeichnet.

11.5.7 Periode innerhalb t1 .. t2

Ein Trigger-Event wird ausgelöst, sobald die **Periodendauer** sich **innerhalb** der beiden Zeiten **t1 und t2** befindet.



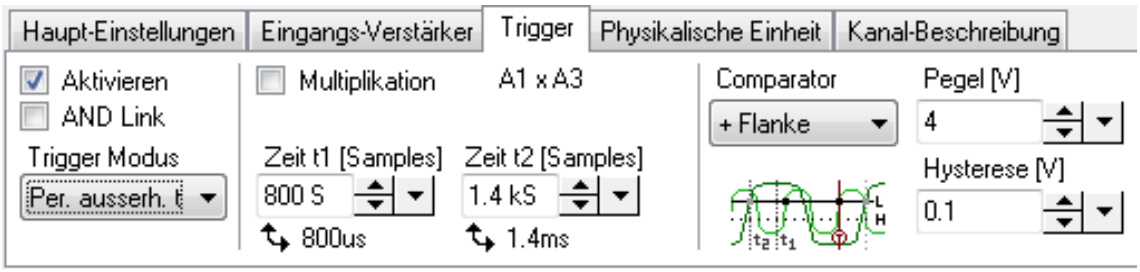
Einstellungen für Kanal 1.



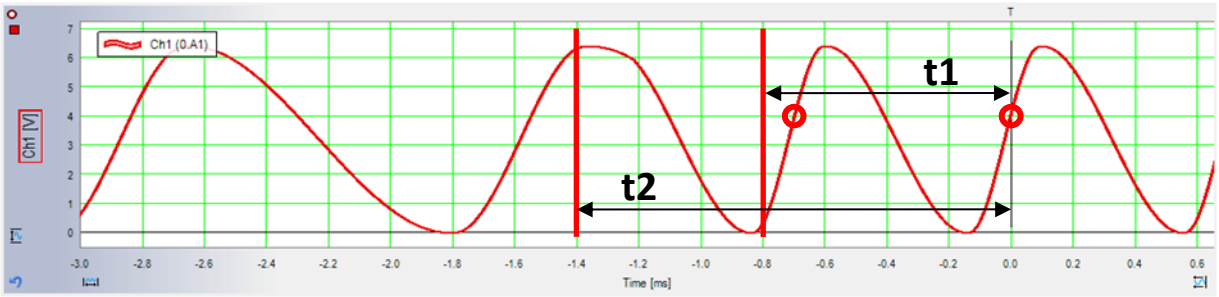
In diesem Bild ist die Periodendauer ca.1.0ms (d.h. innerhalb von 0.8.und 1.4ms).

11.5.8 Periode ausserhalb t1 .. t2

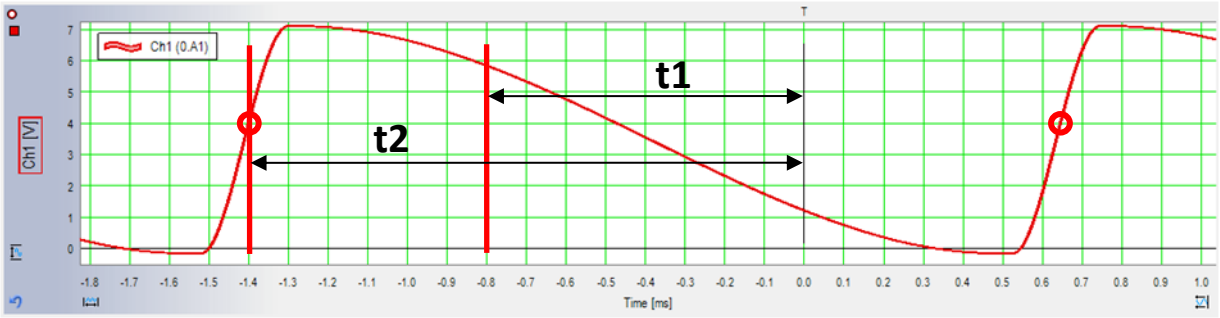
Ein Trigger-Event wird ausgelöst, wenn sich die **Periodendauer ausserhalb** der beiden Zeiten **t1** und **t2** befindet



Einstellungen für Kanal 1.



*In diesem Bild ist die Periodendauer **<t1**.*

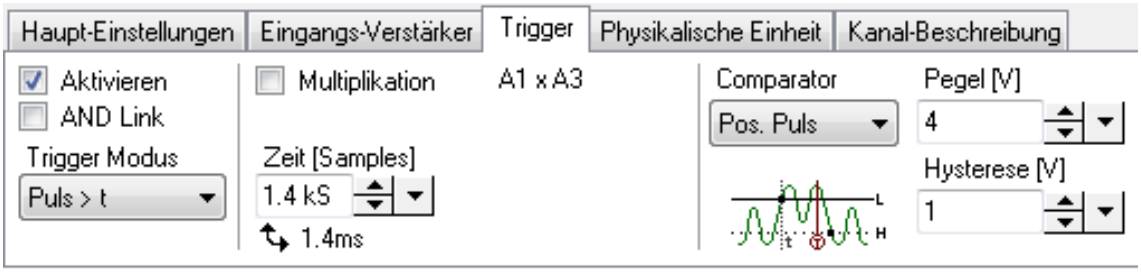


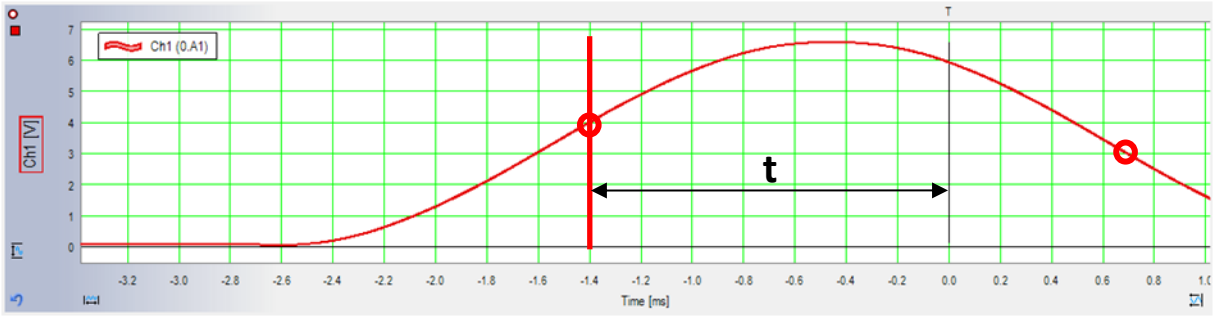
*In diesem Bild ist die Periodendauer **>t2**.*

Der Trigger-Zeitpunkt befindet sich genau **t2** hinter dem Kriterium für den Beginn einer Periode (hier pos. Flanke, Pegel = 4V). Das Signal-Verhalten zeigt also genau beim Trigger-Nullpunkt keinen Trigger-Hinweis. Das Ende der Periode (hier pos. Flanke bei 4V) befindet sich dahinter, eventuell. sogar im Unendlichen oder wird je nach Block-Länge nicht mehr aufgezeichnet.

11.6 Bisherige Trigger-Modi bezogen auf Puls/Periode

11.6.1 Puls > t

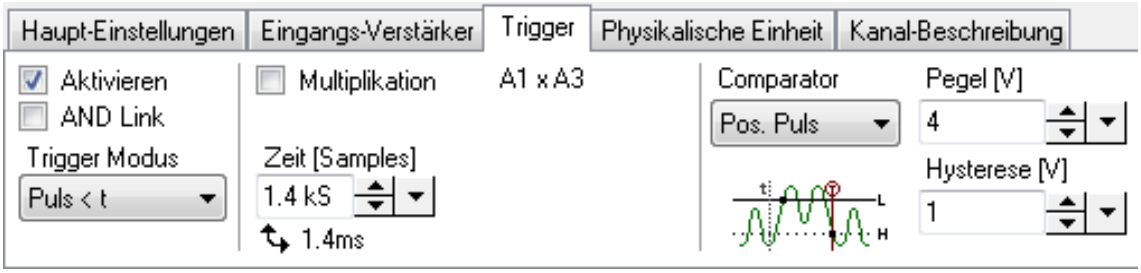


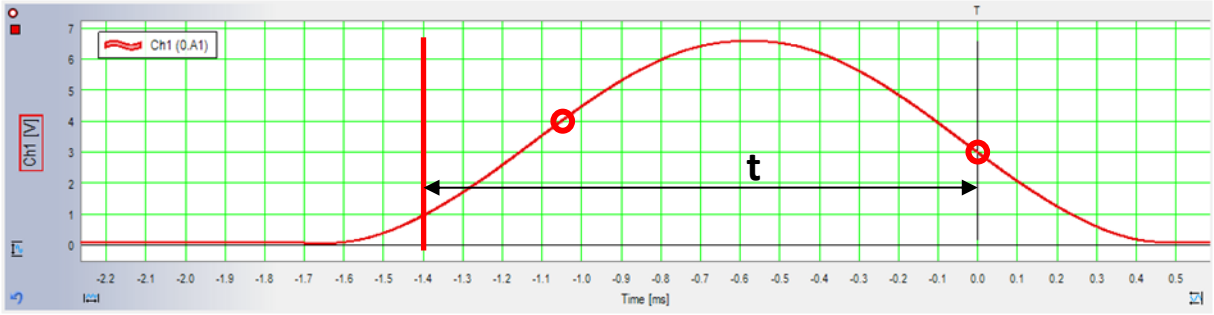


*In diesem Bild ist die **Pulsbreite > t**.*

Der Trigger-Zeitpunkt befindet sich genau um die Zeit **t** hinter dem Kriterium für den Beginn des Pulses (hier pos. Flanke, Pegel = 4V). Das Signal-Verhalten zeigt also genau beim Trigger-Nullpunkt keinen Trigger-Hinweis. Das Ende des Pulses (hier neg. Flanke bei 3V) befindet sich dahinter, ev. sogar im Unendlichen oder wird je nach Block-Länge nicht mehr aufgezeichnet.

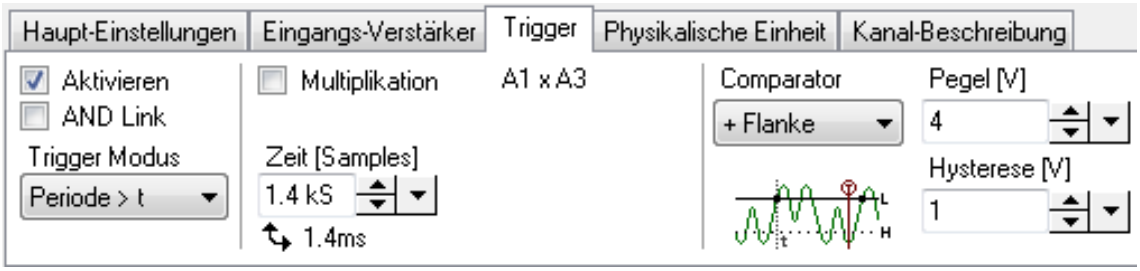
11.6.2 Puls < t

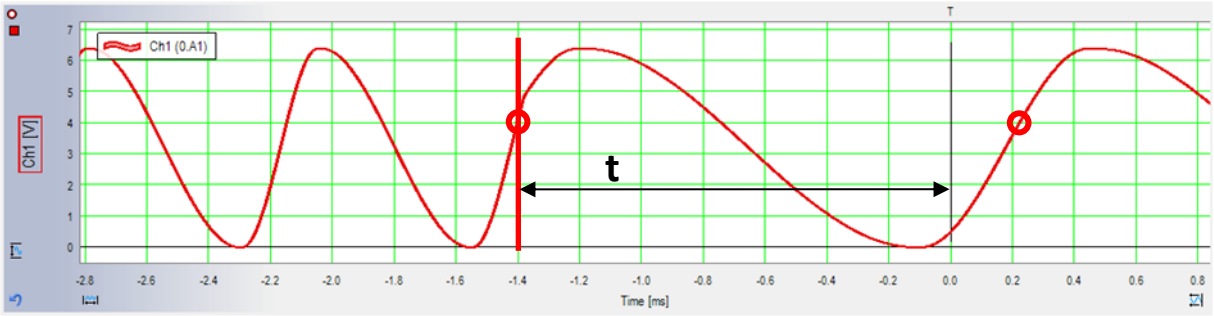




*In diesem Bild ist die **Pulsdauer < t**.*

11.6.3 Periode > t

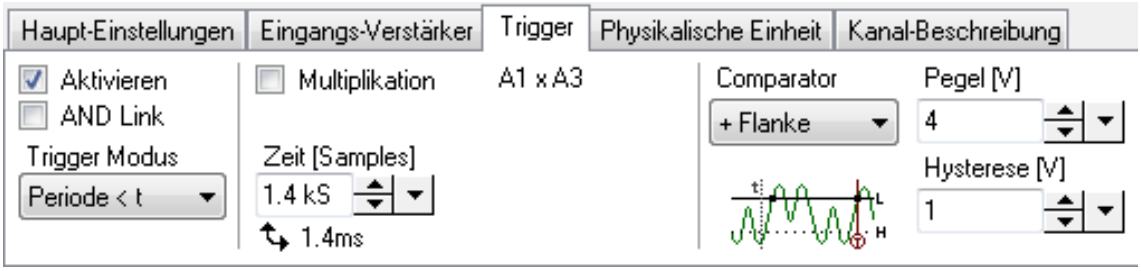


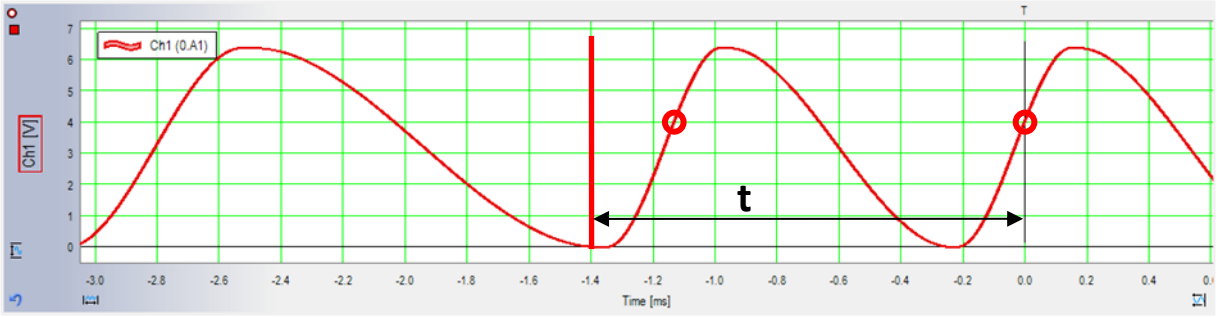


*In diesem Bild ist die Periodendauer **> t**.*

Der Trigger-Zeitpunkt befindet sich genau um die Zeit **t** hinter dem Kriterium für den Beginn einer Periode (hier pos. Flanke, Pegel = 4V). Das Signal-Verhalten zeigt also genau beim Trigger-Nullpunkt keinen Trigger-Hinweis. Das Ende der Periode (hier pos. Flanke bei 4V) befindet sich dahinter, ev. sogar im Unendlichen oder wird je nach Block-Länge nicht mehr aufgezeichnet.

11.6.4 Periode < t

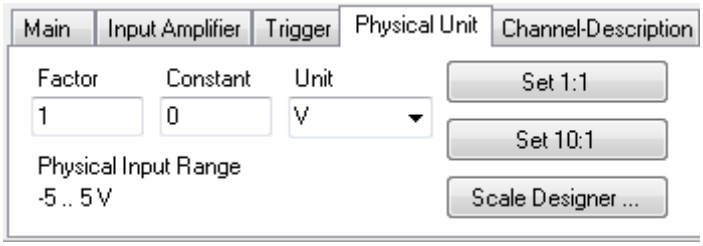




*In diesem Bild ist die Periodendauer **< t**.*

12 Physikalische Einheit

TranAX kann die **gemessenen Kurven** in eine frei wählbare **physikalische Einheit umrechnen** (lineare Transformation).



In dieser Registerkarte können für jeden Kanal individuell eingestellt werden:

- **Faktor und Konstante** für die Umrechnung
- die **physikalische Einheit**

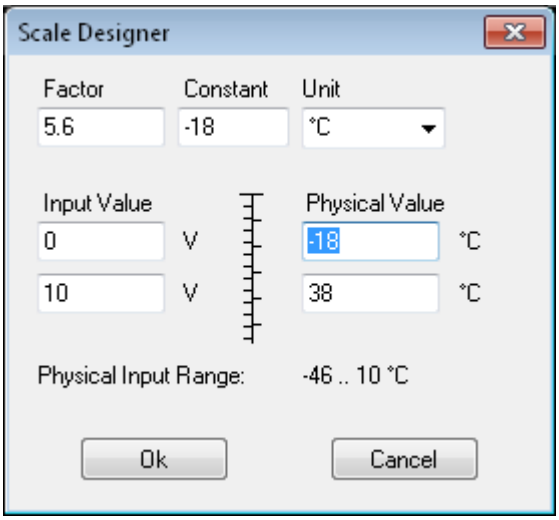
Die Umrechnungsformel lautet:

Physikalische Grösse = (Gemessener Wert [V] * Faktor) + Konstante [Physikalische Einheit]

Die Einstellungen für die Umrechnung müssen vor der Aufnahme eingegeben werden.

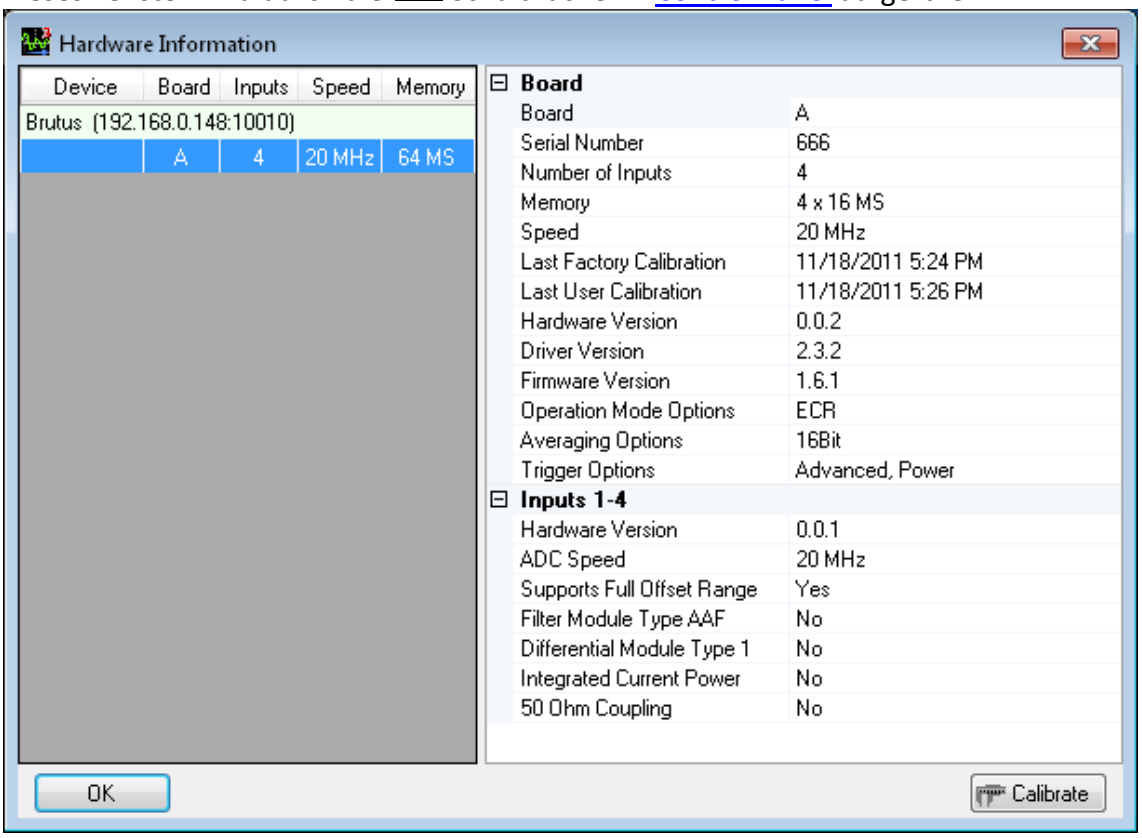
12.1 Scale Designer

Um den Faktor und die Konstante **komfortabel** aus **zwei Referenzpunkten** zu ermitteln, kann der **Scale Designer** verwendet werden, welcher mit der entsprechenden Schaltfläche aufgerufen werden kann.



13 Informationsfenster

Dieses Fenster wird durch die Schaltfläche im Control Panel aufgerufen.



Es zeigt die **aktuell verwendete Hardware** an, mit der TranAX gerade verbunden ist. Diese kann entweder eine oder mehrere auf dem lokalen Rechner eingebauten TPCX/TPCE-Module sein, oder auch ein externes Gerät wie ein TraNET EPC oder TraNET FE. Im Falle eines Fehlers wird eine entsprechende Meldung angezeigt.

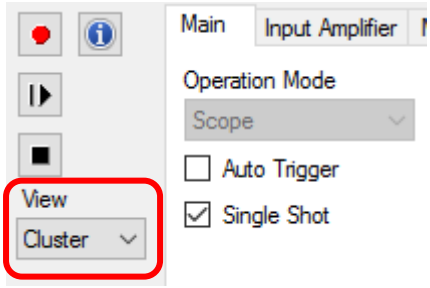
Um die Mess-Hardware zu kalibrieren, kann der Knopf Kalibrieren gedrückt werden. Der Vorgang dauert einige Sekunden.

Die Hardware sollte erst kalibriert werden, wenn diese ihre **Betriebstemperatur** erreicht hat. Eine Kalibration im kalten Zustand kann zu späteren Ungenauigkeiten bei den Messungen führen!
Um die Kalibration zu starten, müssen laufende Aufnahmen erst gestoppt werden.

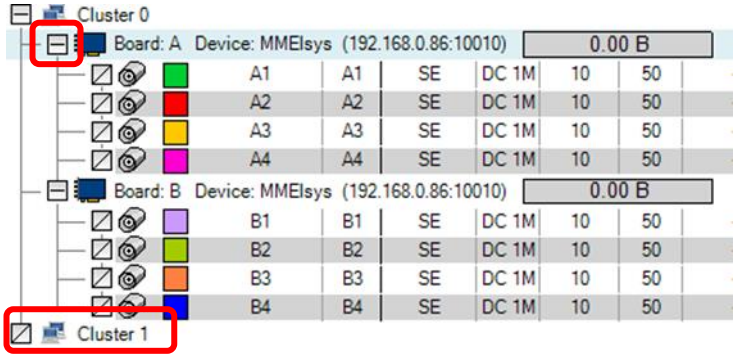
Die Kalibration kann jederzeit wiederholt werden.

14 Cluster Konfiguration

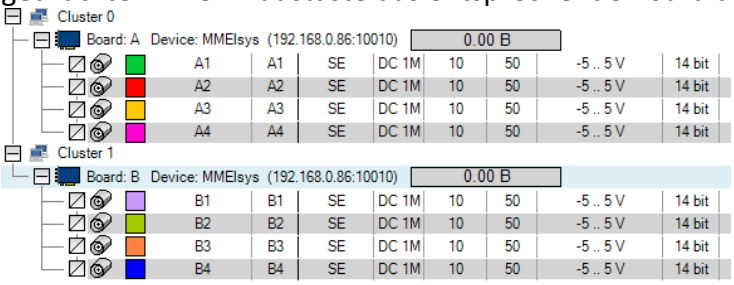
Cluster dienen dazu, **Gruppen von Datenerfassungs-Kanälen zu definieren**, die mit der gleichen Konfiguration arbeiten. Mit Hilfe der Cluster können unterschiedliche Abtastraten und Betriebsarten eingestellt werden. Um Cluster zu erstellen, muss auf die Clusteransicht gewechselt werden:



Es können verschiedene TPCX/TPCE-Gruppen mit der Cluster Konfigurationsmaske definiert werden. **Anfangs sind alle TPCX**/TPCE-**Module im Cluster 0**. Um ein Modul zu einem Cluster hinzuzufügen, erstellt man zuerst einen Cluster indem man einen Rechtsklick in der Geräteansicht tätigt und dann auf **„Cluster hinzufügen“** in Dropdown Menu klickt. Somit wurde soeben eine neue Cluster-Node erstellt:



Diesem Cluster kann jetzt via Drag & Drop ein Board hinzugefügt werden. Dafür zieht man mit gedrückter linker Maustaste das entsprechende Board auf den **Cluster 1:**



Jeder Cluster hat seine eigene Betriebsart und individuelle Einstellungen (Abtastrate, Block Länge, Trigger-Verzögerung etc.)

Alle Cluster arbeiten **synchron** betreffend **Systemtakt** und **Triggerlogik**. Falls ein Cluster ein Trigger-Ereignis auslöst, werden **alle anderen Cluster** auch auf dieses **Ereignis reagieren**, egal in welcher Betriebsart sie sich befinden.
Darauf sollte geachtet werden, wenn mit verschiedenen Betriebsarten gearbeitet wird (z.B. würde ein Trigger im Single-Shot-Cluster auch den kontinuierlicher-Mode mit Stopp-Trigger beenden).

15 Referenzen

In TranAX ist es möglich mit sogenannten Referenzkanälen zu arbeiten. Dies hat den Vorteil, dass eine Referenz auf eine einfache Art und Weise ausgetauscht werden kann, ohne dass man den Kanal zuerst löschen muss. Eine Referenz kann einen Hardwarekanal oder eine Tpc5 Datei enthalten.

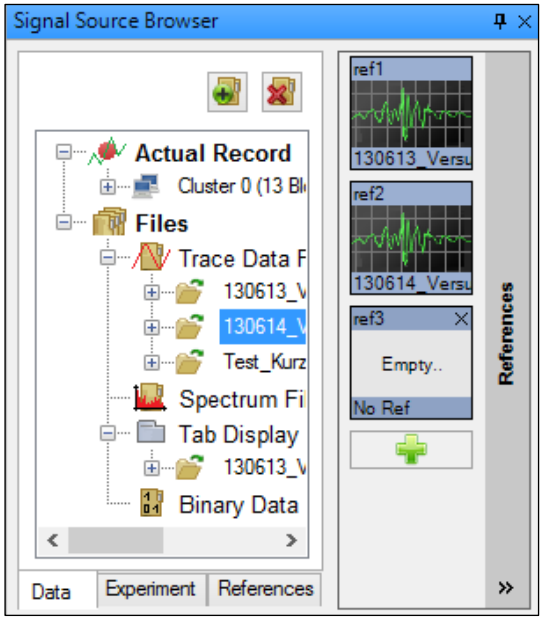
Mit den Referenzen hat man die Möglichkeit einzelne Tpc5-Dateien, welche man beispielsweise als Messreihe mit einer Autosequenz erzeugt hat, nacheinander zu analysieren. Eine Referenz kann man sich als Platzhalter vorstellen, welcher in einer Kurvenanzeige, in den Skalar-Tabellen oder im Formeleditor verwendet werden kann. Beim Zuordnen von Kurven aus einer anderen Datei (oder direkt aus Hardware-Kanälen) werden nur die Daten ausgetauscht, die Eigenschaften (Farbe, Strichdicke, Zoombereich, Labels usw.) bleiben jedoch erhalten.

Dies bietet den **Vorteil**, dass das **Ausarbeiten** und **Vorbereiten** eines **Layouts**, d. h. Kurvenanzeigen, Skalar-Tabellen, platzierte Texte usw. **nur einmal mit den Referenzkurven erstellt werden muss**. Anschliessend können andere Messkurven via Drag & Drop zugewiesen werden, ohne diese im Layout oder in einer Formel jedes Mal ersetzen zu müssen.

Es werden keine Kopien der Dateien erstellt. Die Referenz verhält sich wie ein Zeiger, welcher auf die gewählte Signal-Quelle (Datei oder Hardware-Kanäle) verweist.
Wird eine solche **Quelldatei** auf dem Rechner **entfernt** oder **verschoben**, so verschwindet auch die Referenz, d. h., sie wird leer.

15.1 Mögliche Quellen für Referenzen

Die Handhabung der Referenzen geschieht im Signalquellen-Browser ( ).



Im Signalquellen Browser findet man gleich neben dem Datei-Baum die Referenz-Felder vor. Diese können mit dem vertikalen grauen Balken zur Ansicht gebracht werden.
Durch Klicken auf das Symbol können weitere (vorerst leere) Felder hinzugefügt werden.
Es lässt sich nur das jeweils unterste Feld entfernen. Man klicke dazu auf das kleine "x" oben rechts im Feld.
Um nun eine Referenz zu belegen, zieht man das gewünschte Objekt (eine gesamte Datei oder auch nur einzelne Kurven) via **Drag & Drop** auf ein Referenz-Feld.
Normalerweise wird danach dieses Referenz-Feld via **Drag & Drop** in eine Kurvenanzeige gezogen.

Folgende Objekte sind gültig:

- **Gespeicherte Dateien**, dabei werden die folgenden Formate unterstützt:
  - ***.Tpc5**: TranAX 3 Standard Format für gemessene Kurven im Zeitbereich
  - ***.TPS5**: TranAX 3 Standard Datenformat für Spektrum und FFT Kurven
  - ***.TDP**: gespeicherte Tab-Pages in TranAX 3
  - ***.BDF**: binäres Rohdatenformat von Schreiber- und ECR-Mode
- **Hardware Kanäle**, alle Kanäle aus dem Control Panel bzw. aus einem aktuellen Cluster, dabei können auch mehrere selektiert und dann via Drag & Drop aus den ersten beiden Kolonnen im Control Panel auf ein Referenz-Feld gelegt werden.
- **Gemittelte YT- und FFT-Kurven** (@- bzw. %-Kurven), hier können ebenfalls mehrere Kanäle selektiert und dann auf ein Referenz-Feld gezogen werden.

Die belegten Referenz-Felder können dann ebenfalls via Drag & Drop in eine Kurvenanzeige gezogen werden. Dies betreffen immer all die Kurven, welche auf die entsprechende Referenz gelegt wurden. Wenn also z. B. nur eine Kurve behandelt werden soll, muss auch nur diese eine Kurve auf das Referenz-Feld gezogen werden.

15.2 Referenzen im Formeleditor

Im Formeleditor kann ebenfalls mit den Referenzen gearbeitet werden.
Mit dem File-Befehl kann auf eine Kurve in einer Referenz zugegriffen werden.

```
trace = File(ref1,index) [.blkNo] ['markerNo]
```

Der Index bezeichnet die Kurve in der Datei (0…)). Anstelle des kompletten Datei-Namens in Anführungs-Zeichen muss nur die Bezeichnung ***ref*** mit der entsprechenden Nummer angegeben werden.

Falls auf die erste (nullte) Kurve in einer Referenz zugegriffen werden muss, kann die Formel auch vereinfacht werden:

```
trace = ref2 [.blkNo] ['markerNo]
```

Diese Anweisung entspricht trace**=File(ref2,0)**

Wie üblich können fakultativ auch Block- und/oder Marker-Nummern angegeben werden.

Folgende Funktionen sind ebenfalls zulässig:

```
val = FileIndexExist (ref1, 2) ; val = True oder False
nTrc1 = NTracesInFile (ref1); ist gleichbedeutend wie Length (ref1)
nTrc2 = Length (ref2)
```

Ref wird im Formeleditor als Schlüsselwort verwendet.

15.3 Referenzen in der Autosequenz

Referenzen können auch in den Autosequenzen verwendet werden. Beim Befehl **Save** kann anstelle des Dateinamens die entsprechende Referenz (**ohne Namenserweiterung** z. B. TPC5 usw.) angegeben werden:

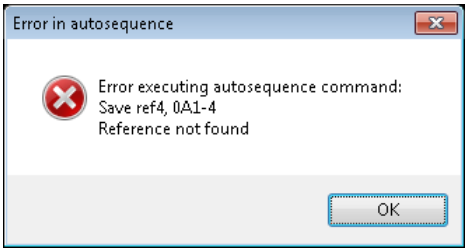
```
Save ref1, 0A1-4
```

Mit dieser Methode ist es möglich in einem Autosequenz-Loop einerseits gemessene Kurven zu archivieren (auf Dateien mit fortlaufender Nummerierung) und gleichzeitig die aktuellen Messdaten für die Anzeige oder die Weiterberechnung im Formeleditor zu verwenden.

Beispiel einer solchen Autosequenz:

```
Repeat 10
Start Recording
Wait on EOR
Save xy_#.tpc5, 0A1-4   ; Erzeugen einer Messreihe
Save ref4, 0A1-4        ; Erzeugen (Überschreiben) einer Datei
                        ; "ref4.tpc5" und Zuweisen zu der
                        ; entsprechenden Referenz
Calculate
Wait for Calculations
Next
```

Der Ausdruck "ref" dient als Schlüsselwort und wird entsprechend geprüft!
Sollte das Referenz-Feld mit der entsprechenden Nummer (z. B. "ref4") im Signal Source Browser nicht vorhanden sein, erscheint eine Fehlermeldung:

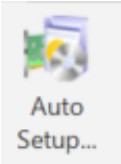


Durch Klicken auf das Symbol  im Signal Source Browser können weitere Referenz-Felder hinzugefügt werden.

In der **Autosequenz** wird die **Kurve ebenfalls physikalisch als Datei** (z. B. "ref1.tpc5") im **Verzeichnis "data" des aktuellen Experiments abgelegt** und der entsprechenden Referenz zugewiesen.

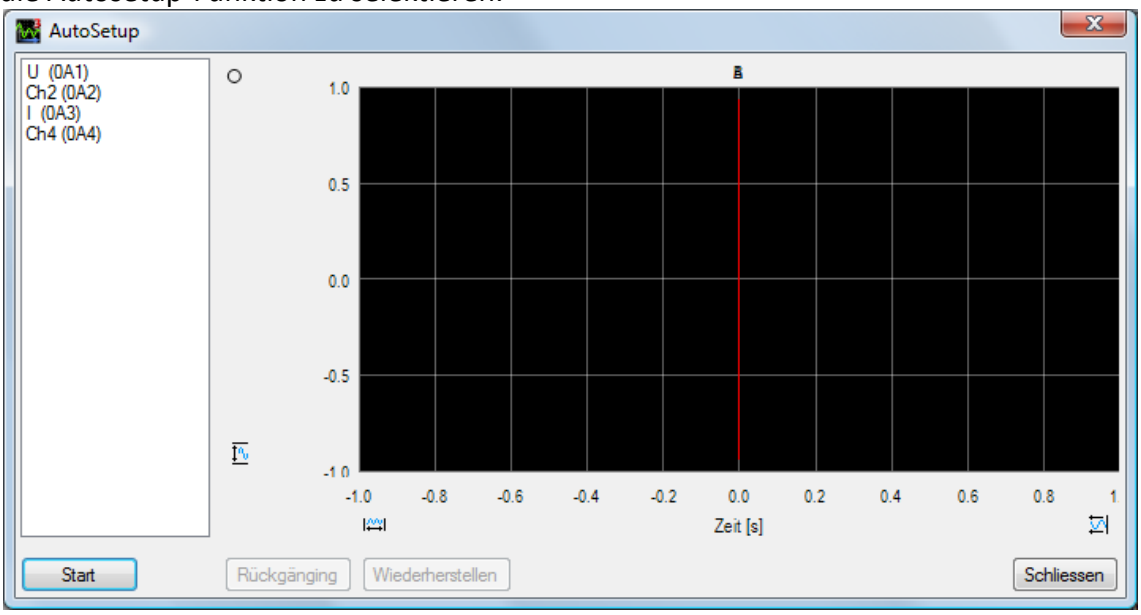
16 Autosetup

Die Funktion Autosetup kann behilflich sein, falls ein **unbekanntes Signal** an den Transientenrekorder angeschlossen wird und ein **schnelles Einstellen** erwünscht ist.
Die Funktion Autosetup passt den **vertikalen Eingangsbereich** an das Signal an, stellt die **Abtastrate auf das Maximum** und die **Blocklänge auf ca. 20 ms**.

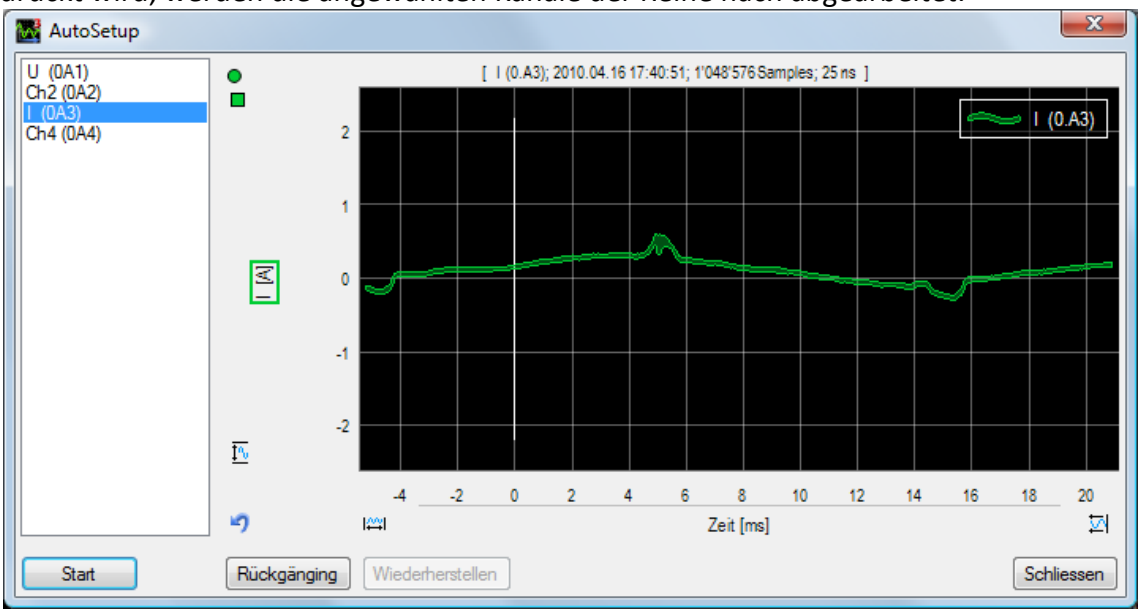


Um den vertikalen Bereich schnell zu finden und das Instrument dazu zu bringen, ein Signal zu erfassen, ohne die Triggerbedingung zu kennen, kann der automatische Setup-Vorgang gestartet werden. Gehen Sie im Ribbon Bar zur Registerkarte "Messung" und klicken Sie auf "Auto-Setup ...".

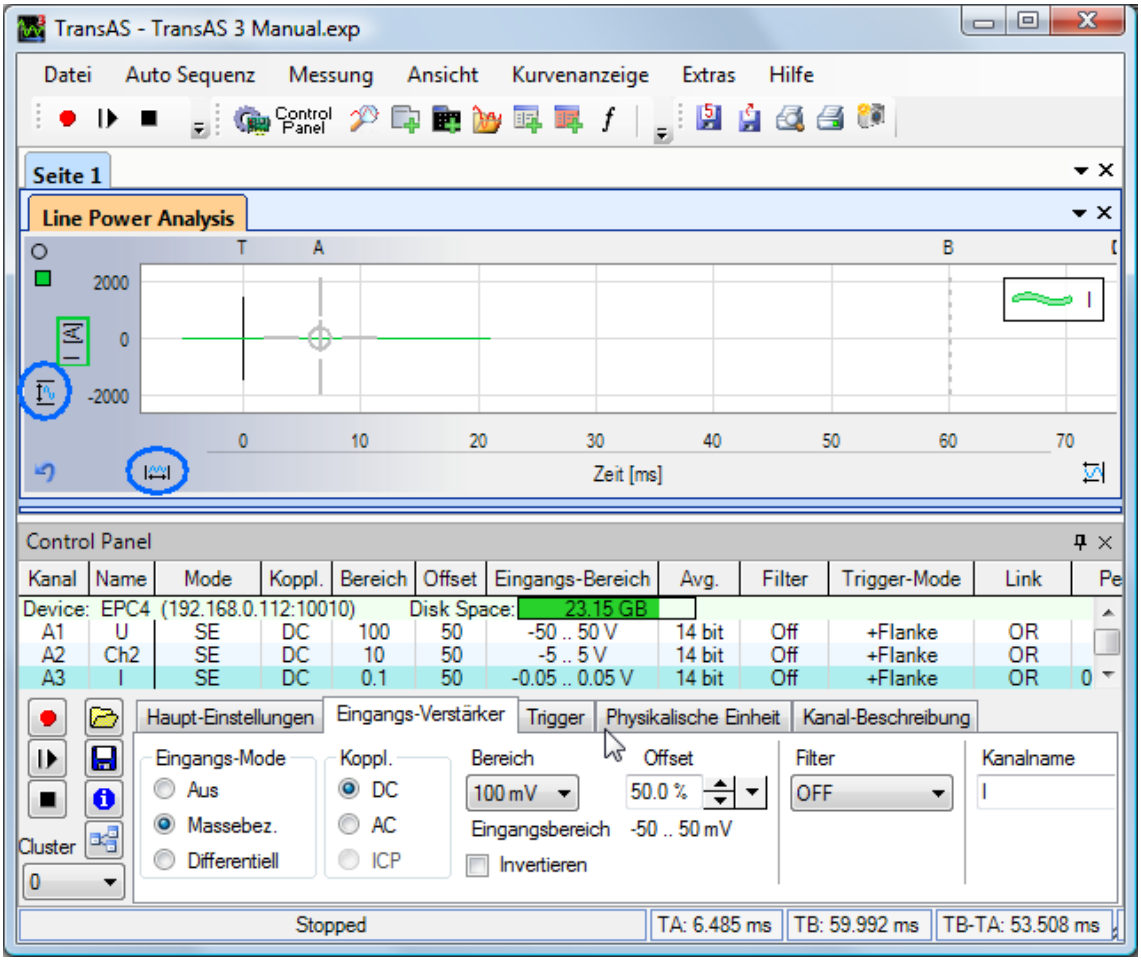
Das **Fenster AutoSetup** erlaubt nun alle zur Verfügung stehenden Kanäle oder einzelne davon für die Autosetup-Funktion zu selektieren.



Ein **einzelner Kanal** wird mit einem **Links-Klick** selektiert. Mehrere Kanäle können mit einem **Ctrl plus Links-Klick** angewählt oder ein- und ausgeschalten werden. Sobald der Knopf "Start" gedrückt wird, werden die angewählten Kanäle der Reihe nach abgearbeitet.



Nachdem Autosetup durchgeführt wurde, wird die Kurve in dem neu ermittelten Messbereich dargestellt. An dieser Stelle, und auch beim erneuten Öffnen dieses Fensters, kann mit einem Klick auf **Rückgängig** der **alte Zustand wiederhergestellt** werden. Falls die neuen Einstellungen übernommen werden sollen, so wird dies mit einem Klick auf Schliessen ausgeführt.

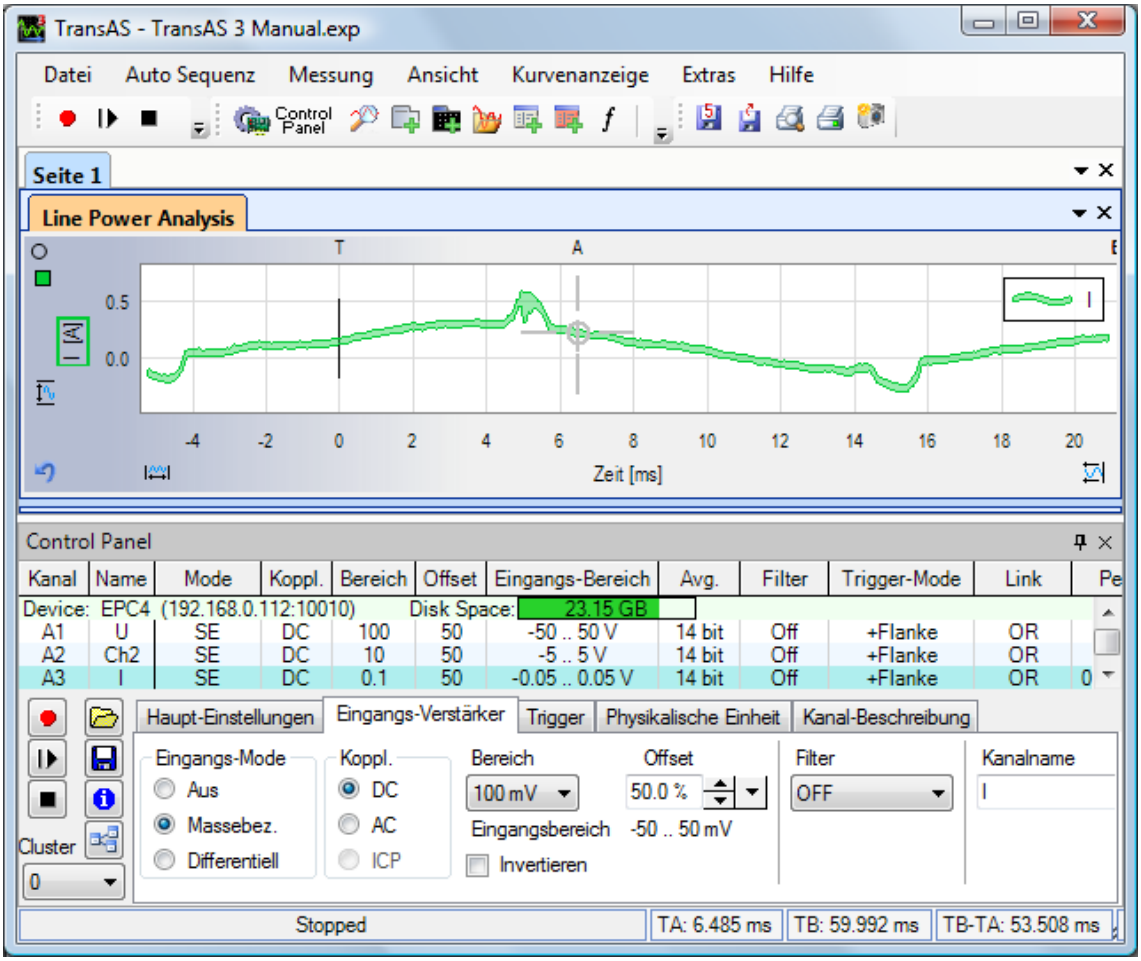


Der Eingangsbereich wurde von 100 V auf 100 mV geändert, die Kurve selbst wird mit derselben Ansicht dargestellt wie zuvor. **Die Kurvenanzeige wird nicht automatisch neu skaliert.**

Durch Klicken der beiden Knöpfe **Volle Ansicht X** und **Volle Ansicht Y kann dies nun manuell durchgeführt werden.**

	Die X-Achse auf volle Ansicht strecken.
	Die Y-Achse auf volle Ansicht strecken.

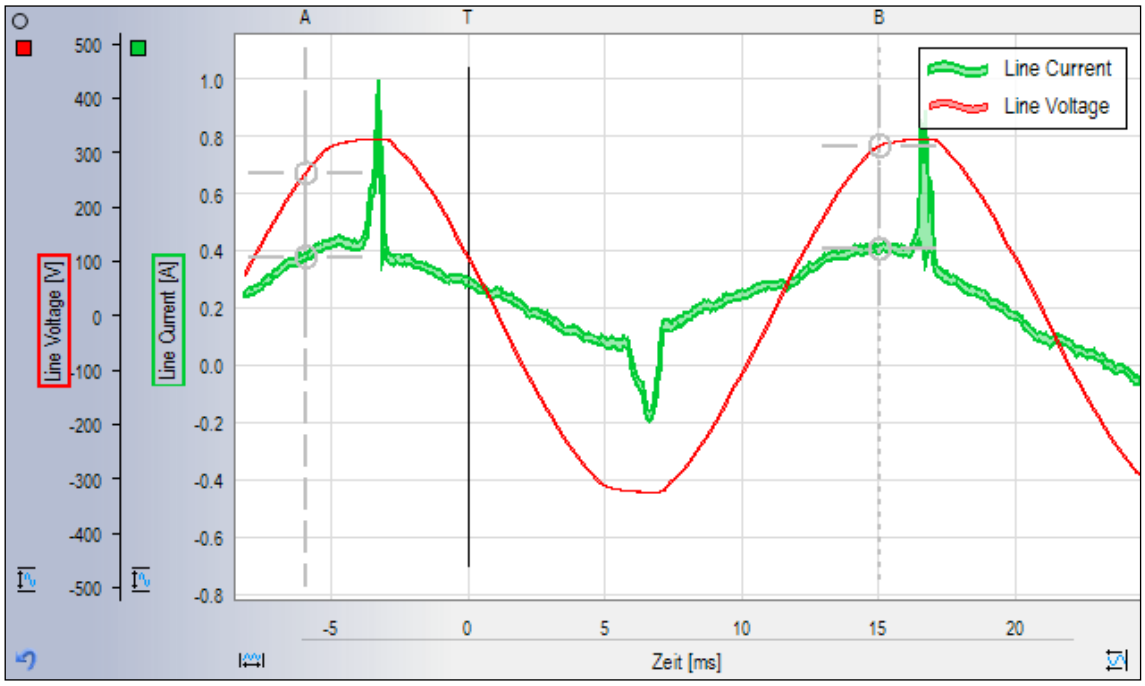
Nachdem die beiden oben erwähnten Zoom-Knöpfe gedrückt wurden, kann man die Details der Kurve erkennen. Die Ansicht kann jederzeit angepasst werden, in dem zum Beispiel näher in die Kurve gezoomt wird.



Mit besserer Kenntnis über das Signal können nun die Trigger-, Vertikal- und Zeitbasis-Einstellungen gezielter vorgenommen werden.
Falls die Triggerquelle eines angewählten Kanales eingeschalten ist, so wird deren Trigger-Mode automatisch auf positive Flanke gestellt und der Pegel auf die Mitte gelegt. Ansonsten werden die Einstellungen für Trigger nicht verändert.

17 Kurvenanzeige (Allgemeines)

Die Kurvenanzeige dient der Visualisierung von aufgenommenen oder laufenden Signalen.

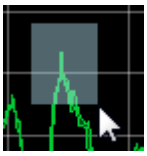


17.1 Organisieren und Platzieren

Mittels Drag & Drop kann ein Kanal vom Control Panel oder vom Signalquellen-Browser in der Kurvenanzeige dargestellt werden. Falls mehrere Signale in einer Kurvenanzeige überlagert werden, erhält jedes Signal **eine bestimmte Farbe** und wird am linken Rand durch ein **farbiges Kästchen repräsentiert**. Jede Kurvenanzeige gehört zu einer Seite. Eine **neue Kurvenanzeigen** wird mittels der Schaltfläche oder im Menü ***"Ansicht" / "Neue Kurvenanzeige"*** hinzugefügt. Um die Kurvenanzeigen neu zu platzieren geht man gleich vor wie mit den Sub-Fenstern: Einfach auswählen und an die gewünschte Stelle verschieben.

Um eine **bessere Übersicht** zu wahren, falls viele Seiten und Anzeigen offen sind, können kann mittels **Rechtsklick** auf eine Seite- oder Kurven-Registerkarte das **Kontextmenü** geöffnet werden. Hier kann man dann die Anzeigen schliessen, einen Titel anpassen oder die Fenster vertikal oder horizontal platzieren.

17.2 Zoomen

	Man kann einen **Bereich vergrössern,** indem ein **Rechteck** über den gewünschten Bereich gezogen wird. Dazu die obere linke Ecke anwählen und mit gedrückter linker Maustaste bis zur unteren rechten Ecke ziehen. Anschliessend die Maustaste wieder loslassen.
	Den Mauszeiger auf die Achsenbeschriftung bewegen bis ein **Zoom-Symbol** erscheint. Anschliessend die linke Maustaste drücken und je nachdem, ob die Maus nach oben/unten bzw. rechts/links bewegt wird, vergrössert oder verkleinert sich der Ausschnitt. Hier kann auch mit dem Scrollrad gezoomt werden.

17.3 Kurven verschieben

Um den Ausschnitt in der Kurvenanzeige zu verschieben, kann innerhalb des Kurvenfensters an einer freien Stelle die **rechte Maustaste gedrückt** und in die gewünschte Richtung bewegt werden.
Andererseits kann der Mauszeiger über die Achseneinheiten platziert werden bis ein doppelseitiger Pfeil erscheint. Danach kann durch Klicken und Bewegen die X- bzw. Y-Achse mitsamt den Kurven verschoben werden. Auch hier kann mit dem Scrollrad gearbeitet werden.

Verschieben des Kurven-Zeitbereichs (X-Skala) mit dem Maus-Rad geht auch, wenn der Maus-Cursor innerhalb des Kurvenfensters ist und gleichzeitig die Shift-Taste gedrückt wird.

17.4 Volle Ansicht

Es gibt zwei Icons, um **die volle Ansicht** wiederherzustellen:

	Zeige Y-Achse in voller Ansicht (für jede Achse einzeln).
	Zeige X-Achse in voller Ansicht.

Mit folgenden Schaltern können Sie **Zoomen und Verschieben rückgängig** machen oder wiederherstellen.

	Die letzten Änderungen in der Kurvenanzeige rückgängig machen (Undo)
	Die letzten Änderungen in der Kurvenanzeige wiederherstellen (Redo)
	Falls im Kontinuierlichen oder ECR Mode **Autoscroll** aktiviert ist, wird die Kurvenanzeige automatisch mit dem Signal **horizontal mitlaufen.** Falls deaktiviert, wird es die Anzeige belassen, aber die Aufnahme läuft weiter!

	Pin Option um Fenster als Registerkarte zu minimieren (Signal Quellen Browser oder Control Panel)

18 Y/T-Kurven-Anzeige-Fenster

Die folgenden Anzeige-Optionen werden mittels **Rechtsklick** im **Kurvenanzeige-Fenster** oder (Auszugsweise) via Menü ***"Kurvenanzeige"*** erreicht.

Verschieden Parameter, wie z.B. Cursors, Gitter, Triggerlinien usw. können zu oder abgeschaltet werden.

Cursor A, B (oder ev. zusätzliche) können in der Kurvenanzeige an der aktuellen Position des Maus-Zeigers platziert werden. Natürlich muss dazu "Cursors" aktiviert sein.

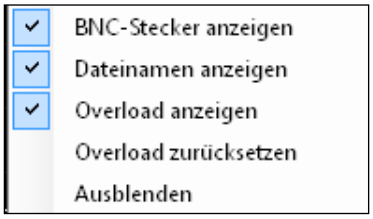
Das Gleiche gilt für die Horizontal-Cursors.

Die Anzahl der Achsen (links und rechts) können auch via Rechts-Klick auf den Y-Achsen selbst definiert werden.

Via "Eintrag hinzufügen" lassen sich (auch nachträglich) an der aktuellen Maus-Position RecordingLog-Einträge vornehmen.
Über "Gehe zu Eintrag" wird der Kurvenanzeige-Ausschnitt direkt beim entsprechenden RecordingLog-Eintrag gesetzt.

18.1 Legende

Die Legende kann ein- oder ausgeblendet werden (Ausblenden). Falls diese nicht mehr sichtbar ist, kann sie jederzeit über das Menü ***"Kurvenanzeige" / "Legende"*** wieder angezeigt werden.

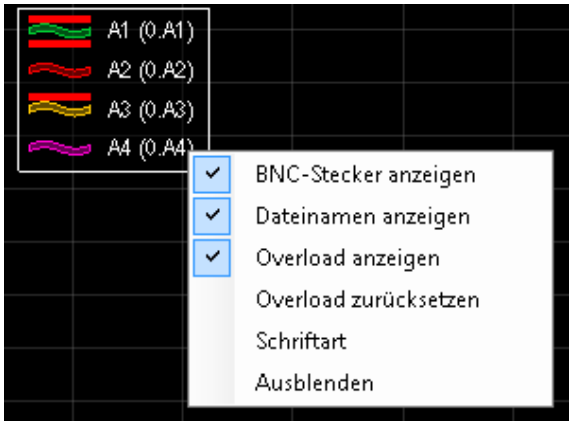


Ein Rechtsklick zeigt das Pop-up-Menü auf der linken Seite. Um die Legende an eine andere Stelle zu verschieben, fahren Sie mit der Maus über die Legende und halten Sie die linke Maustaste gedrückt.

Falls "**BNC-Stecker anzeigen**" nicht aktiv ist, wird nur der Kanalname angezeigt. Falls eine Kurve von einer Datei importiert oder mit dem Formel-Editor berechnet wurde kann der **Dateiname** eingeblendet werden.

Die **Option "Overload anzeigen"** zeigt an, ob der Verstärker des entsprechenden Kanals positiv oder negativ übersteuert wird oder wurde. Dies wird mit einer Linie oberhalb bzw. unterhalb des Kurven-Symbols dargestellt.

Die Überlauf-Anzeige signalisiert mit einer roten Linie eine momentane Übersteuerung. Eine gelbe Linie signalisiert eine Übersteuerung die vorgängig (seit Start der Aufzeichnung) mindestens einmal aufgetreten ist.
Ein registrierter Überlauf (gelbe Linie) kann zurückgesetzt werden, indem auf "Overload zurücksetzen" geklickt wird.



Durch einen Klick auf den Menüpunkt **"Overload zurücksetzen"** werden die gelben Markierungen, welche den Overload signalisieren, oberhalb oder unterhalb der Kurve wieder entfernt.

Das **Zurücksetzen** eines eingetretenen Overload-Ereignisses kann sinnvoll sein, wenn z.B. im Schreiber-Modus lange Aufzeichnungen vorgenommen werden und nach einer Störung oder Kalibration der Sensoren der Overload wieder zurückgesetzt werden soll. So kann bei einer **späteren Kontrolle** sofort erkannt werden, ob **wieder ein Überlauf aufgetreten ist** oder die Signale ohne Störungen aufgezeichnet wurden.

Overload ist das **Übersteuern** des dynamischen Bereiches des **ADCs**. Falls dieser z.B. auf 2V, 50% Offset eingestellt ist, liegt der messbare Bereich zwischen -1V und 1V. Falls nun ein Signalpuls von 1.5V auftritt, würde dieser einen Overload auslösen. Dieser wird zur **Laufzeit als roter Balken** oberhalb (bei negativem Overload unterhalb) der Legenden-Kurve dargestellt. Ein **gelber Balken** signalisiert, dass **vorgängig** ein Über- bzw. Unterlauf **aufgetreten** ist.

Bei jedem "Overload-Zurücksetzen" wird ein **Eintrag im Aufnahmeprotokoll** vorgenommen. Somit sind alle Manipulationen, welche während der Aufnahme vorgenommen wurden, protokolliert. Via Menü ***"Ansicht" / "Aufnahmeprotokoll"*** können die eingetragenen Ereignisse dargestellt werden.

Die **Schriftart der Legende** kann man benutzerdefiniert einstellen. So ergibt sich die Möglichkeit einen kleinen Font zu wählen, damit die Legende möglichst schlank bleibt.

18.2 Text-Eintrag in Kurvenanzeige-Fenstern

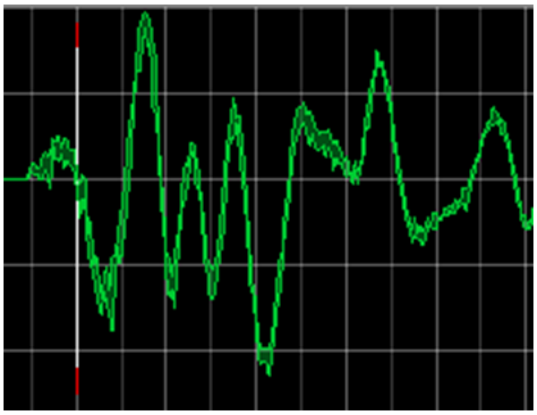
Via "Text" lassen sich Einträge in den Kurvenfenstern vornehmen. Mit Rechts-Klick über einem Text-Eintrag kann danach auch die Schriftart angepasst werden.
Bei der Texteingabe in den Kurvenanzeige-Fenstern können auch Verknüpfungen zu berechneten Einzelresultaten im Formel-Editor eingefügt werden. Dabei müssen die Resultat-Namen als Solche in spitzigen Klammern geschrieben werden. Beispiel: **<VarName>.** Anstelle von *VarName* muss der Resultat-Name geschrieben werden. Falls dieser nicht existiert oder wenn noch keine Berechnung durchgeführt wurde, wird "NotDefined" angezeigt. Vor und nach **<VarName>** kann ein beliebiger Text stehen. Wenn *VarName* ein Zahlen-Resultat ist, kann es sein, dass dieses mit zu vielen Stellen geschrieben wird. Um dies zu vermeiden kann mit der Formel **StringFormat()** die Zahl in einen String gewandelt werden (z.B. xyzStr=StringFormat(xyz,"0.00". Text im Fenster: "... <xyzStr> ...").

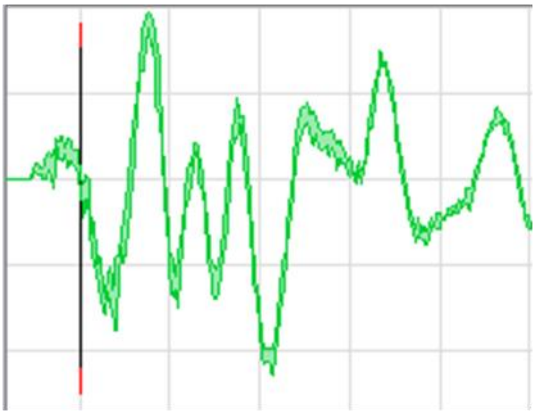
18.3 Gitter

Beim Gitter kann zwischen einem groben und einem feinen Raster gewählt werden. Falls gewünscht kann man dieses auch ganz ausschalten.
Unter dem Menü "Extras/Einstellungen/Benutzeroberfläche..." kann die Farbe des Gitters vorgewählt werden.

18.4 Hintergrundfarbe

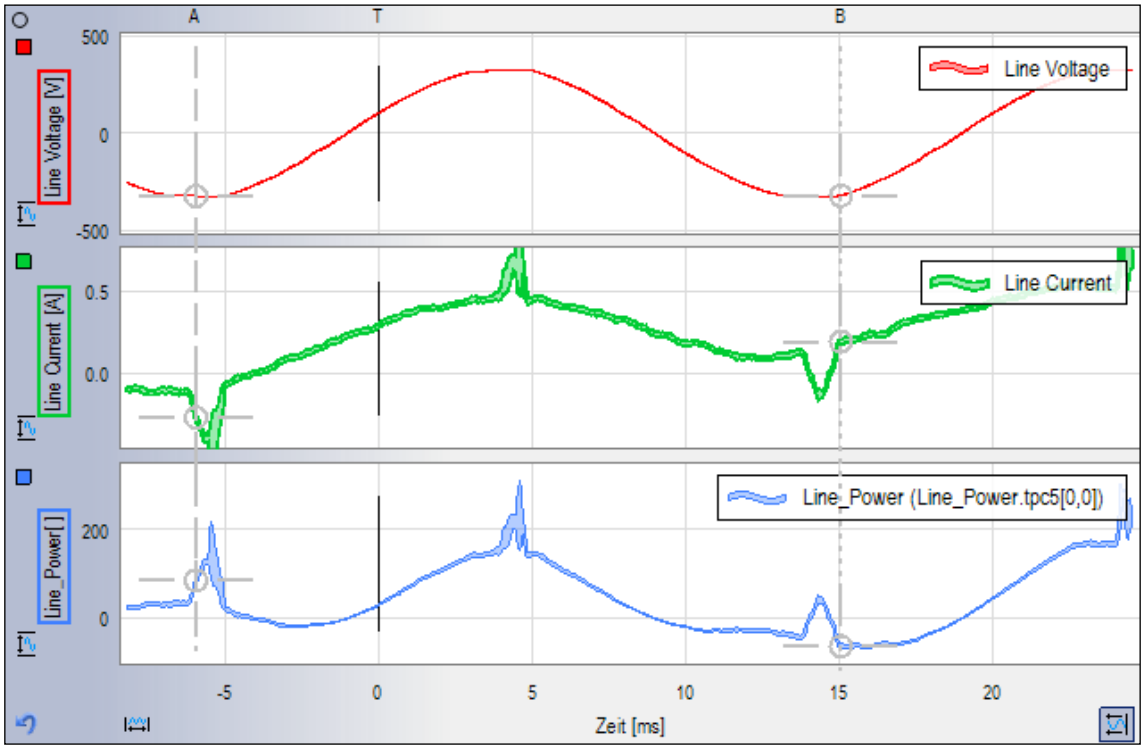
Die Hintergrundfarbe der Anzeige kann man je nach gewünschtem Kontrast entweder auf schwarz oder weiss stellen.
Auch die Hintergrundfarbe kann via Menu "Extras/Einstellungen/Benutzeroberfläche..." vorgewählt werden.





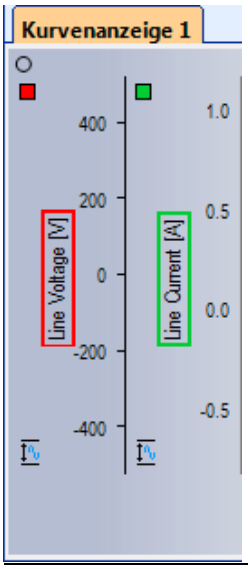
18.5 Bereiche

Anstatt Kurvensignale überlappend anzeigen zu lassen, können diese auch parallel in einzelnen Bereichen dargestellt werden. Es können bis zu 16 Bereiche pro Kurvenanzeige-Fenster eingestellt werden.



18.6 Y-Achsen

18.6.1 Anzahl Y-Achsen links, rechts



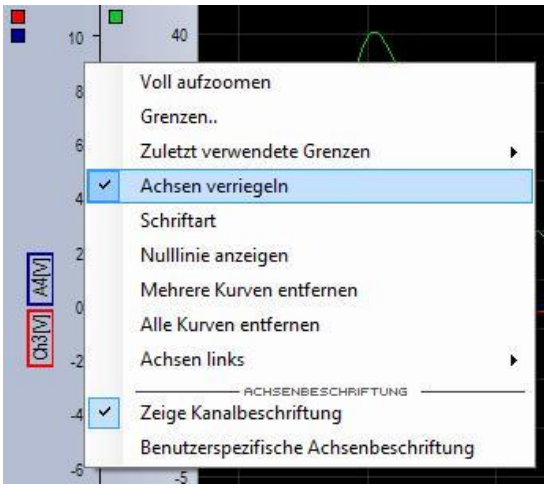
Es können für die einzelnen Signale jeweils **separate Y-Skalen** definiert werden. Durch Rechts-Klick in der Kurvenanzeige, "Achsen links" bzw. "Achsen rechts" kann man die Anzahl Skalen bestimmen. Dies kann auch via Menü "Kurvenanzeige" / "Skalen auf der linken Seite" bzw. "Skalen auf der rechten Seite" gemacht werden.

Nachdem zusätzliche Skalen hinzugefügt wurden, können die Kurven mit den farbigen Signal-Kästchen links von der Kurvenanzeige auf die neuen Skalen gezogen werden.
Die einzelnen Y-Skalen lassen sich dann individuell Zoomen und Verschieben.

Es können bis zu 12 Y-Achsen links und rechts erstellt werden.

Man kann der Kurvenanzeige auch vorgängig neue Skalen hinzufügen und anschliessend die Signalkurven direkt vom Control Panel oder Signalquellen-Browser auf die gewünschten Skalen ziehen.

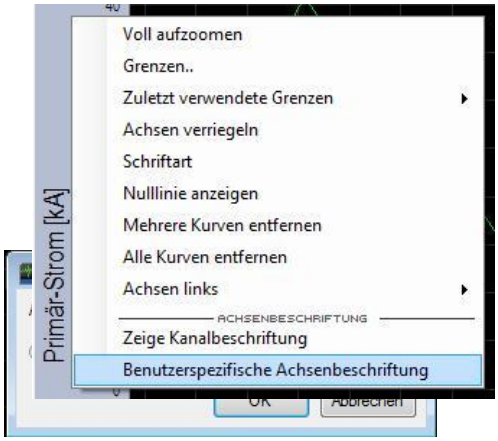
18.6.2 Y-Achsen verriegeln



Wenn in Kurvenanzeige-Fenstern **mehrere Y-Achsen** deklariert wurden, können diese miteinander **verriegelt** werden.
Mit einem Rechts-Klick auf einer der Y-Achsen öffnet man das Pop-up-Menü und wählt *"Achsen verriegeln"*.
Falls man dann jeweils zoomt oder die Kurven verschieben möchte, werden alle Achsen diese Aktion ausführen.

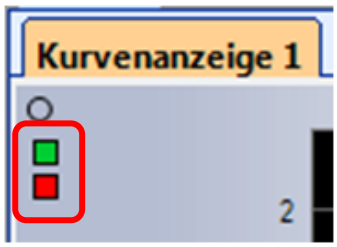
18.6.3 Y-Achsen Beschriftung

Die Beschriftung der Y-Achsen kann auch benutzerspezifisch erfolgen:



Um die angepassten Kurvenanzeige-Einstellungen zu speichern, wählen Sie im *"Datei" / "Layout speichern.."*. In den Layout-Dateien wird die Anordnung von sämtlichen Fenstern und deren inneren Eigenschaften gespeichert.

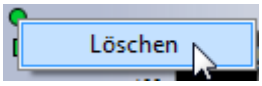
18.7 Kurvendarstellung



Jede Kurve lässt sich in der Kurvenanzeige oben auf den Y-Achsen als kleines **Kästchen** identifizieren. Bei ausgefülltem Kästchen ist die Kurve sichtbar, ein **leeres Kästchen** macht die Kurve **unsichtbar**.
Durch Rechts-Klick auf einem Kästchen kann eine Kurve aus dem Fenster entfernt werden. Zudem können verschiedene Kurveneigenschaften eingestellt werden.
Um die **Reihenfolge** der Kästchen zu ändern, einfach die **Maustaste gedrückt halten** und an einer Position weiter unten loslassen.

Wenn eine Kurve von der **Kurvenanzeige entfernt** wurde, bedeutet dies nicht, dass die Kurvendaten gelöscht wurden! Die Kurven-Eigenschaften (Farbe, Labels, Y-Skalierungen usw.) gehen jedoch verloren.
Um diese Kurve wieder darzustellen, kann sie einfach wieder mittels Drag & Drop vom Control Panel oder vom Signalquellen-Browser in die Kurvenanzeige oder auf eine Y-Achse gezogen werden.

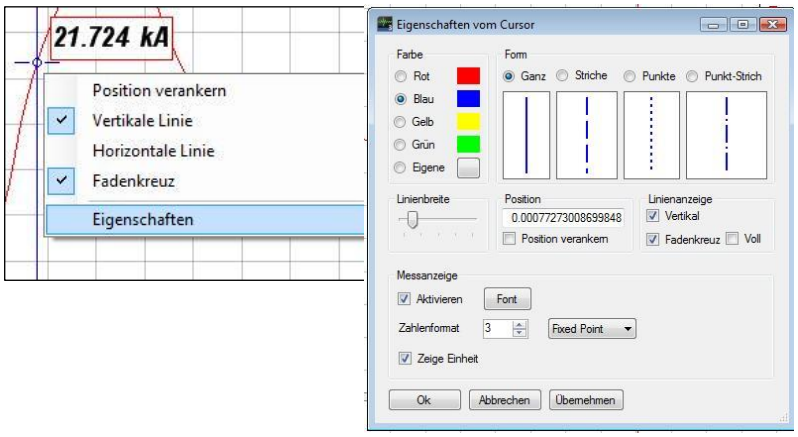
Jede Kurvenanzeige hat einen **Platzhalter** für eine **Statuszeile**, die ein- oder ausgeblendet werden kann mit dem kleinen **runden Knopf** in der linken oberen Ecke des Fensters. Diese Status-Zeile enthält die wichtigsten Informationen von einem beliebigen Signal in der Kurvenanzeige. Mit **Drag & Drop** von einem beliebigen Kurven-Kästchen werden die Informationen dieses Signals in der Statuszeile angezeigt.
Die Statuszeile kann nützlich sein bei der Generierung von Rapporten (via Drucken oder Snapshot), da wichtige Informationen wie Instrument-Name, Kanalname, Zeitstempel, Anzahl Samples und das Abtastintervall festgehalten sind.

	Drag vom Kanal-Schalter …
	… und Drop auf der Statuszeile oberhalb der Kurve.
	Rechtsklick auf den Runden Knopf zeigt die Option, um die Statuszeile zu löschen

Falls die Statuszeile nicht mehr erwünscht ist, kann sie entweder mit Links-Klick auf dem runden Knopf ausgeschalten oder per Rechtsklick/Löschen gelöscht werden.

18.8 Cursor-Eigenschaften

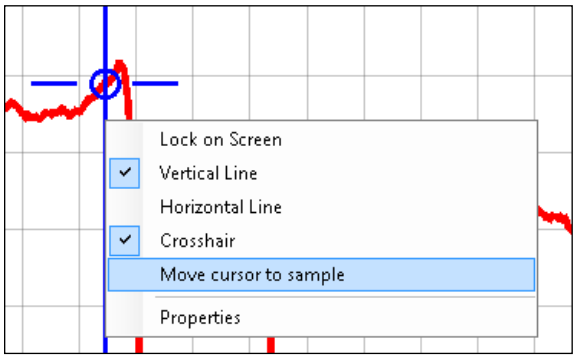
Wenn das Maus-Symbol über einen Cursor platziert wird, kann mit Rechts-Klick die Eigenschaften-Maske geöffnet werden.
Darin können das Aussehen und weitere Eigenschaften des Cursors eingestellt werden
Wenn die "*Messanzeige*" aktiviert ist, werden für jede Kurve die Y-Werte an der Cursor-Position angeschrieben.



Im Menu "Extras -> Einstellungen" hat man in der Rubrik "Benutzeroberfläche" die Möglichkeit, den Cursor Buchstaben (A, B, C …) oben innerhalb der Kurvenanzeige oder auf der Statuslinie darüber darzustellen (die Statuslinie kann via dem kleinen Kreis-Symbol oben links, unsichtbar geschalten werden).

18.9 Cursor auf Abtastpunkten

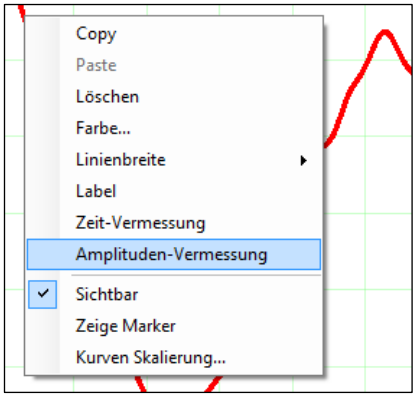
Die Cursors können individuell so eingestellt werden, dass diese nur noch auf den einzelnen Samples bewegt werden können.
Mit einem Rechtsklick auf den Cursor wählt man im Kontextmenu "Bewege Cursor auf Abtastpunkten".



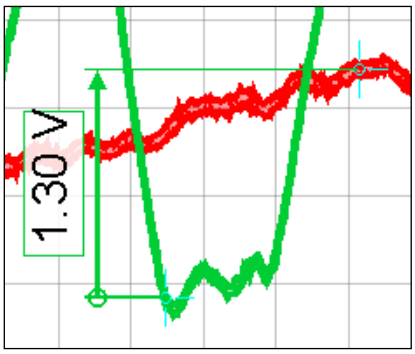
Danach springt der Cursor von Sampling zu Sampling. Zwischenwerte existieren nicht oder allenfalls nur, wenn mehrere Kurven mit unterschiedlichen Abtastraten im gleichen Kurven-Fenster sind.

18.10 Vermessung von Kurven

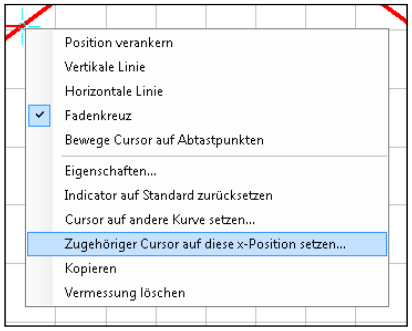
Neben den Labels können auch **Zeit- und Amplituden-Vermessungen** auf die Kurven platziert werden.



Mit einem Rechtsklick auf die entsprechende Kurve erhält man das dazugehörige Kontextmenu.



Hat man die Vermessung platziert, kann man die Cursors in X-Richtung **auseinanderziehen**. Es wird jeweils die Differenz zwischen Cursor 2 und Cursor 1 in einem Textlabel dargestellt. Der Text und das Zahlenformat kann wie gewohnt mit einem Rechtsklick auf den Text eingestellt werden.



Um einen die beiden Cursors auf eine **andere Kurve zu setzen**, tätigt man einen Rechtsklick auf den jeweiligen Cursor und wählt im Kontextmenu "Cursor auf andere Kurve setzen..". Danach erscheint ein Dialog, in welchem man das gewünschte Signal auswählen kann.

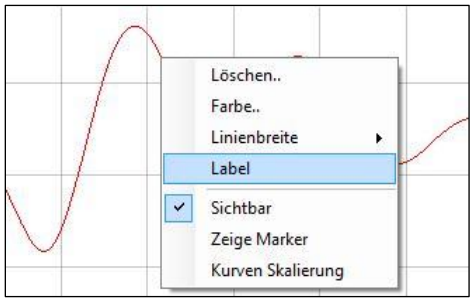
Damit lassen sich Amplitudendifferenzen von zwei Signalen ausmessen (dies macht in der Regel nur Sinn, wenn beide Signale die gleiche Einheit haben).

Via Rechtsklick auf "Zugehöriger Cursor auf diese x-Position setzen…" wird der andere Cursor auf die gleiche X-Position gesetzt.

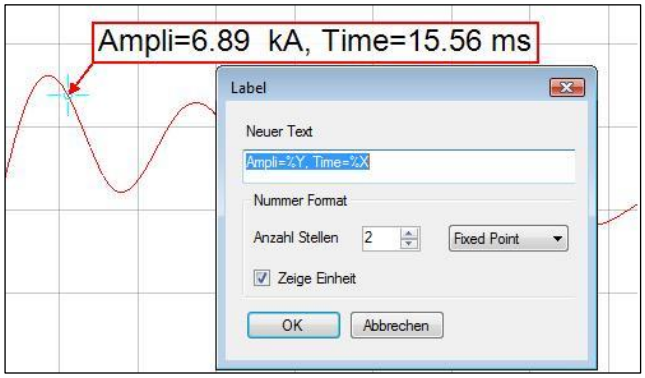
Wenn das Icon ("Cursor miteinander bewegen") gedrückt wird, verschieben sich die Mess-Cursors gemeinsam. Danach lässt sich die Differenz der beiden Kurven an einer beliebigen Stelle auf der X-Achse anzeigen.

18.11 Labels auf Kurven

Die Kurven können mit sogenannten "Labels" angeschrieben werden. Dazu geht man mit dem Mauszeiger an die Stelle auf der Kurve, an der das Label platziert werden soll. Mit Rechts-Klick und Klick auf "Label" wird die Kurve per Default mit dem Amplitudenwert in deren Farbe angeschrieben.

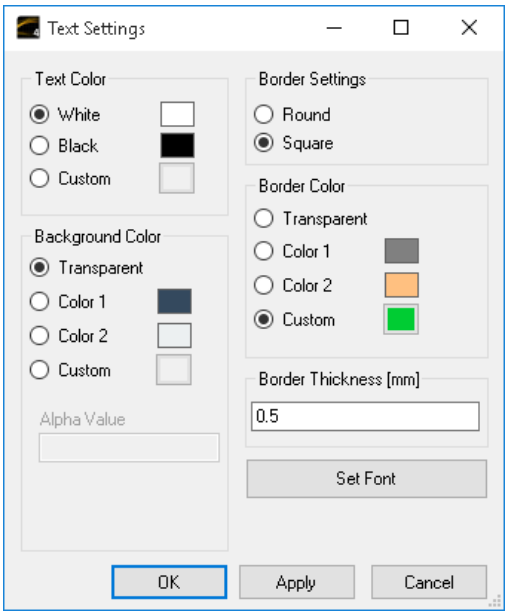


In der Maske, welche sich via Rechts-Klick innerhalb des Labels und Klick auf "Text bearbeiten..." öffnet, kann ein beliebiger Text eingegeben werden.



Die Schlüsselbezeichnungen "%Y" bzw. "%X" werden durch die Werte der Y- bzw. Zeit-Achse ersetzt. Zudem können hier die Zahlen-Eigenschaften der Y- bzw. X-Werte definiert werden.

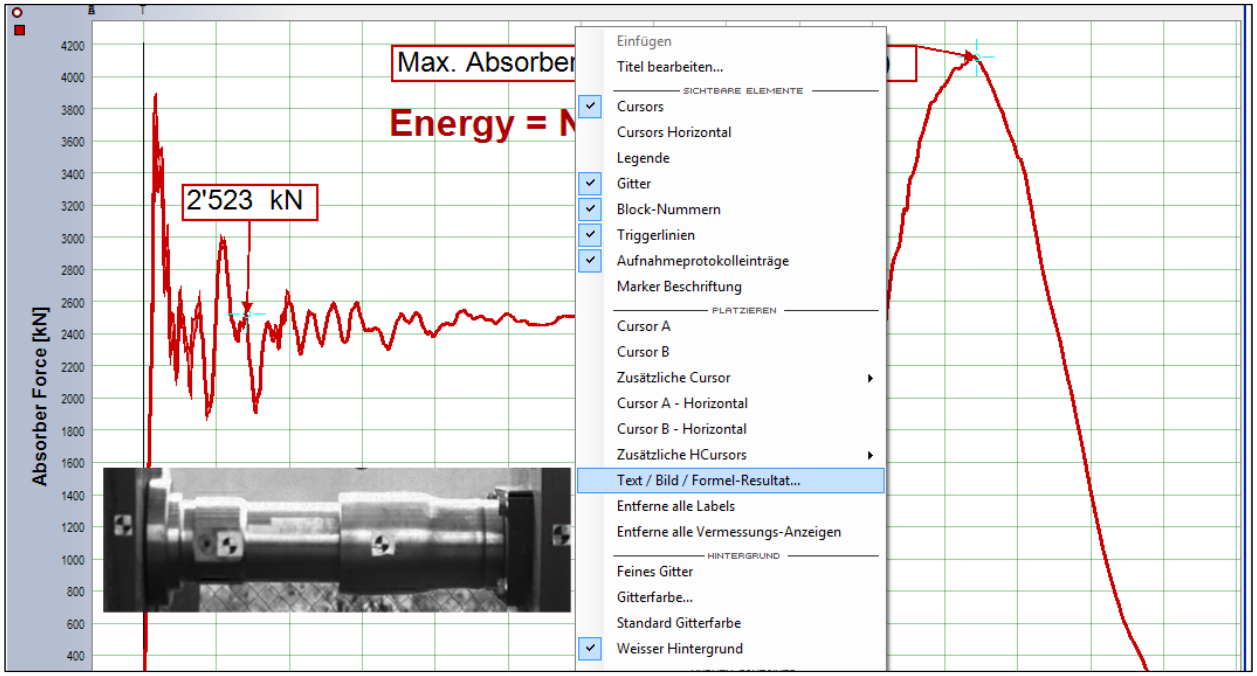
Die Eigenschaften des Label-Fadenkreuzes auf der Kurve können ähnlich wie bei einem Cursor definiert werden. Via das Fadenkreuz kann das Label auf der Kurve wie ein Cursor verschoben werden.



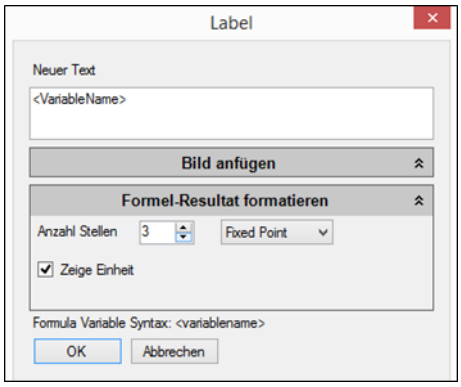
Mit einem Rechts-Klick auf das Label, lässt sich bei "Set Settings..." die Eigenschaften des Labels Anpassen. Farben für Text, Hintergrund und Rahmen können so angepasst werden.

18.12 Bilder einfügen, Resultate aus Berechnungen formatieren

Via Dropdown-Menu (Rechtsklick in einem Kurvenfenster), kann (wie ein Text) auch ein Bild eingefügt werden.



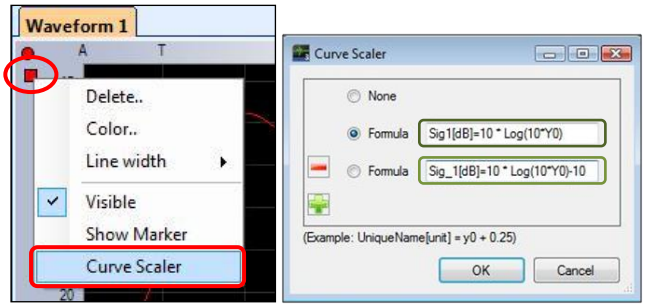
Durch Klicken auf ***Bild anfügen*** kann das gewünschte Bild gesucht und ausgewählt werden. Beim Auswählen wird normalerweise zuerst der Ordner *images* im aktuellen Experiment präsentiert. Durch Navigieren kann jedoch auch ein Bild irgendwo ausgelesen und angefügt werden.



In der gleichen Maske kann auch das Zahlenformat eines, via Formel-Editor berechneten, Resultates festgelegt werden

18.13 Y-Skalierung von Kurven

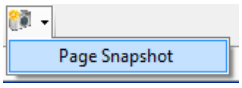
Individuell kann jede Kurve durch eine Formel umgerechnet werden.



Die Formel kann vom Anwender frei **ge**wählt werden. Es lassen sich auch vorgängig mehrere Funktionen definieren. Die gewünschte Formel wird dann durch Anschalten des entsprechenden Knopfes aktiviert.
Diese Formeln gehören dann als Eigenschaft zu der Kurve (wie die Farbe). Beim Verschieben bzw. Kopieren (via Drag & Drop) in ein anderes Fenster werden auch die Skalierungs-Formeln mitgeführt.
Es sind natürlich nur Berechnungen gültig, die sich auf Amplituden-Werte (keine X-Achsen-Werte) beziehen. Auch bezieht sich die Umrechnung nur auf die Anzeige der Kurve in einem Kurvenfenster und einer allfälligen Berechnung in einer Skalar-Tabelle. **Speichern von Kurven erfolgt immer unskaliert**.
Bei Funktionen in einer Skalar-Tabelle zu einer derartigen Kurve wird die Y-Skalierung (mit Name und Einheit) auch berücksichtigt.
In den Bereichen, wo die Formel keine Resultate liefern kann (z.B. bei LOG von negativen Werten), wird keine Kurve gezeichnet.
Die Berechnung erfolgt aufgrund der Rohwerte ***y0*** einer Kurve**.** Als Rohwerte gelten auch allenfalls umgerechnete Werte, welche möglicherweise via Formel-Editor entstanden sind oder welche durch entsprechende Einstellungen unter *Physikalische Einheit* im Control-Panel bereits vor der Aufnahme definiert wurden.
Der neue Kanal-Name (ChName[Unit]= ...) darf nicht aus einem Schlüsselwort des Formel-Editors (z.B. Cos, Sin, If, for, to, as, Pi usw.) bestehen.
Fehler in der Skalierungs-Formel werden beim Eingeben mit gekennzeichnet. Bei fehlerhaften Formeln wird die Berechnung unterdrückt. Der Rohwert bleibt stehen.

18.14 Snapshot

Durch Links-Klick über der Schaltfläche wird das Bild der aktuellen Kurvenanzeige in die Windows-Zwischenablage kopiert und kann von dort in eine Benutzer-Anwendung (z.B. in ein Report-Dokument) eingefügt werden.



Wenn mit der Maus auf den Pfeil rechts des Icons geklickt wird, kann der Bildschirm-Inhalt von allen Fenstern in einer Seite (nicht nur von einem Kurven- bzw. Skalar-Fenster) in die Zwischenablage kopiert werden.

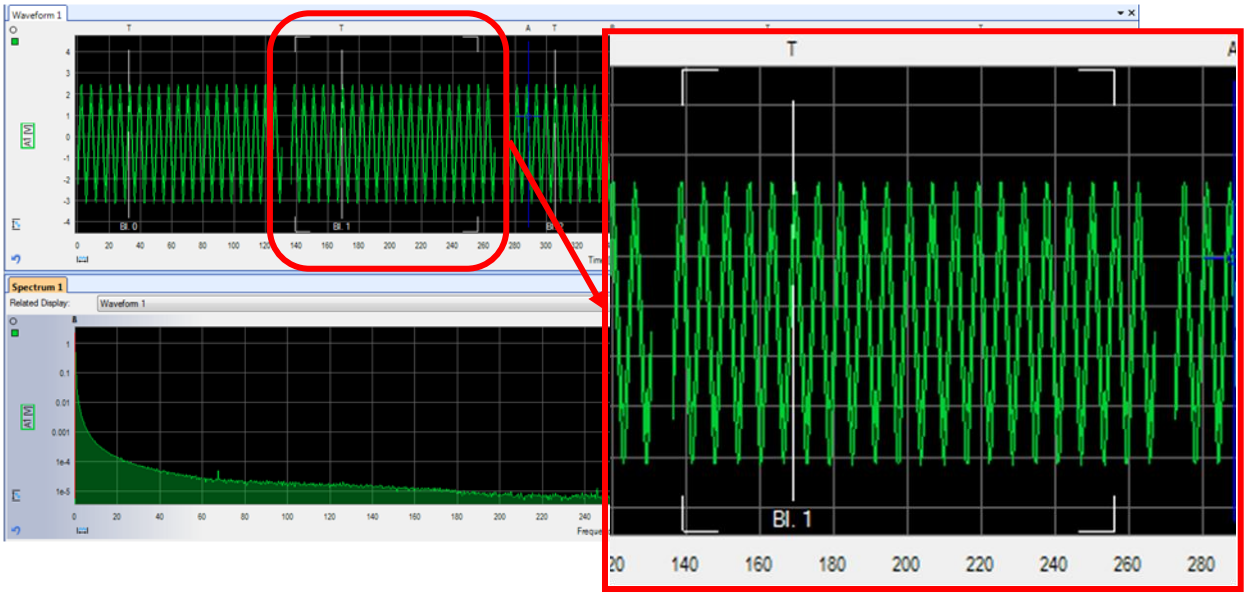
Unter "Extras / Einstellungen / Benutzeroberfläche / Snapshot" kann zwischen "Bitmap" und "Vektor" gewählt werden.
Zudem kann die Grösse, wie der Bildausschnitt danach erscheinen soll, gewählt werden. Bei mehreren Fenstern in der Seite werden diese proportional angepasst. Alle Text-Objekte werden unverzerrt übertragen. Dies sollte beachte werden, wenn in einer Kurvenanzeige benutzerspezifischer Text eingetragen wurde. Dieser kann dann anders platziert oder gar abgeschnitten erscheinen.
Hier kann auch eingestellt werden, dass das Bild immer mit weissem Hintergrund in der Zwischenablage abgelegt wird. Falls dabei auf dem Bildschirm mit schwarzem Hintergrund gearbeitet wird, sollten die Kurven-Farben eher dunkel gewählt werden.

18.15 Analyse von Multiblock-Aufnahmen (Block-Jumping)

Möchte man nach einer Multiblock- oder ECR-Aufnahme von Block zu Block springen, kann man dazu über die Toolbar folgende Steuerelemente verwenden:

Man erteilt dem gewünschten Fenster den Fokus und klickt auf das jeweilige Steuerelement. Anstatt über die Toolbar die Steuerelemente zu bedienen, kann man auch ganz bequem mit den Tasten "Page UP", "Page DOWN" arbeiten. Mit den Tasten "Home“ und "End“ springt man jeweils auf den ersten bzw. den letzten Block.

Falls ein Zweitfenster (XY, Marker, FFT, Zoom) den Fokus hat und dann von Block zu Block gesprungen wird, bewegen sich im Hauptfenster nur die entsprechenden Zeit-Marken

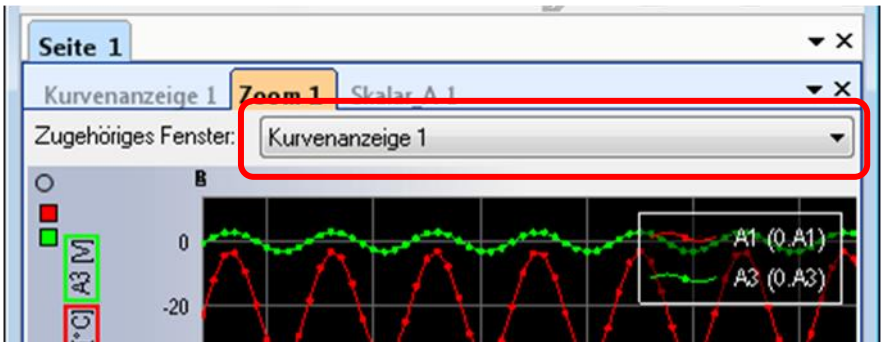


18.16 Zusätzliche Kurvenanzeigen

Neben der regulären YT - Kurvenanzeige gibt es noch **zusätzliche Arten** von Anzeige-Fenstern. Sie werden über das Ribbon Tab **"Layout"** aufgerufen:

- Neue **Zoom-Kurvenanzeige**: Um einen Bereich vergrössert darzustellen.
- Neue **XY-Kurvenanzeige**: Zeigt eine XY-Darstellung an.
- Neue **Marker-Kurvenanzeige**: Zeigt die digitalen Marker Signale.
- Neue **FFT-Kurvenanzeige**: Um das Frequenz-Spektrum des Signals anzuzeigen.

Durch Hinzufügen eines dieser Kurvenanzeige Fenster, können Signalkurven gleichzeitig mit unterschiedlichen Einstellungen dargestellt werden. Eventuell muss das zugehörige Kurvenanzeige-Fenster bestimmt werden (siehe Bild).

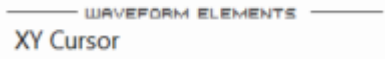
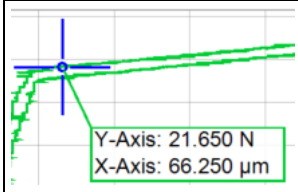
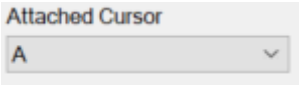


Die zugehörige Kurvenanzeige dient hauptsächlich zur Festlegung von Begrenzungs-Marken für den Zeitausschnitt, auf welche sich die Signale in den speziellen Anzeige-Fenstern beziehen.
Eventuell müssen zusätzlich gewünschte Kurven via Drag & Drop aus dem Control Panel oder dem Signalquellen-Browser geholt werden. Die Kurven können aber auch direkt via Drag & Drop aus dem ursprünglichen Fenster kopiert werden.
In der zugehörigen Kurvenanzeige erscheinen nun vier weisse Ecken (Zeitfensterbegrenzungen), die den Bereich aufspannen, der für das hinzugefügte Kurvenfenster relevant ist. Diese Markierungen können auch mit der Maus verschoben werden.
Für eine Zoom-Kurvenanzeige heisst das, dass der (im zugehörigen Fenster) markierte Bereich vergrössert dargestellt wird.
Bei einer FFT-Kurvenanzeige wird das Spektrum aus dem markierten Bereich berechnet und angezeigt. Analoges gilt für die XY- und Marker-Darstellung.

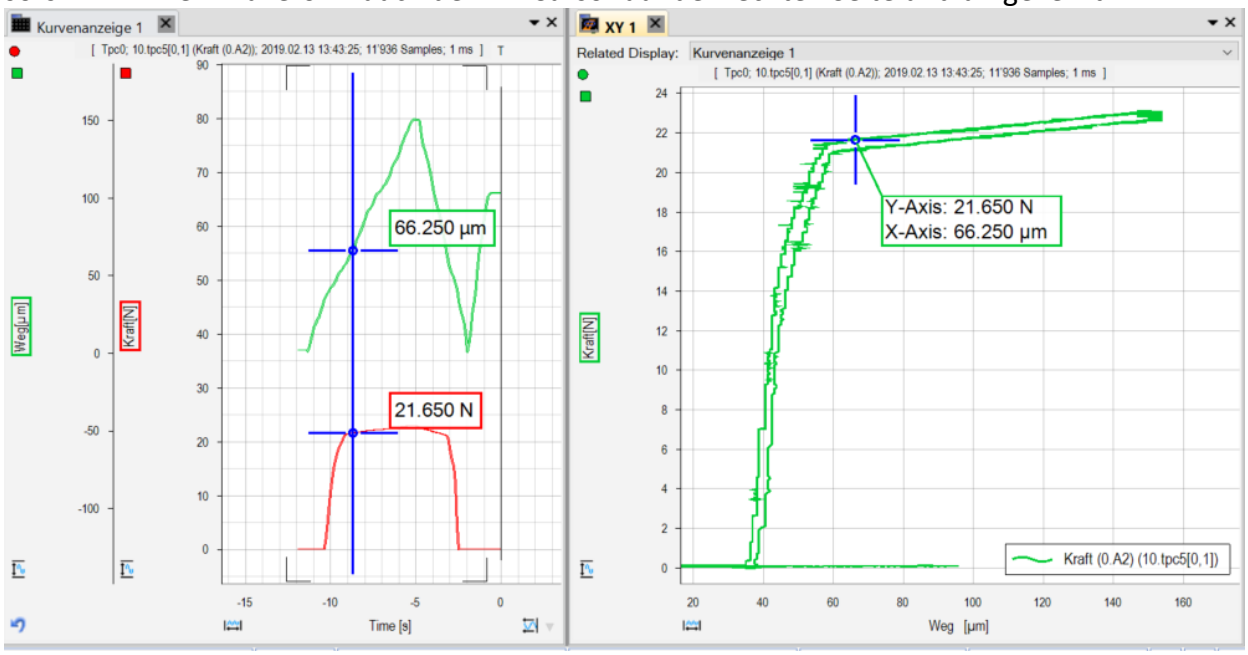
Die zusätzliche Kurvenanzeige-Fenster können auch (wie alle Fenster und Tabellen) an beliebigen Seiten an den anderen Fenstern angedockt werden (Siehe dazu auch das Kapitel TranAX Übersicht).

19 XY Waveform

Das implementierte XY-Waveform ermöglichen das Synchronisieren der Daten des YT-Waveform zusammen mit der XY-Waveform selbst.

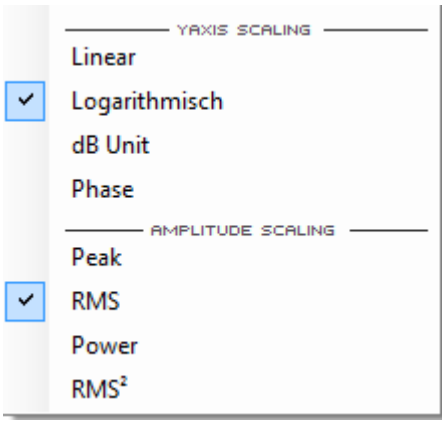
	Rechtsklick auf einen XY Trace ermöglicht das Platzieren eines XY Cursors. Dieser kann dann entsprechend auf dem XY Messkurve umhergezogen werden.
	Dieser Cursor zeigt standardmässig die Werte der X- und Y-Achse.
	Rechtsklick auf den Cursor, dann linksklick auf "Eigenschaften" öffnet einen Dialog, in dem dann der XY-Cursor mit dem YT-Cursor im "Zugehörigen Fenster" gekoppelt werden kann.

Wenn der Cursor in der YT- und XY-Wellenform gekoppelt ist, verschiebt das Bewegen des Cursors A im linken Waveform auch den XY-Cursor auf der rechten Seite und umgekehrt.



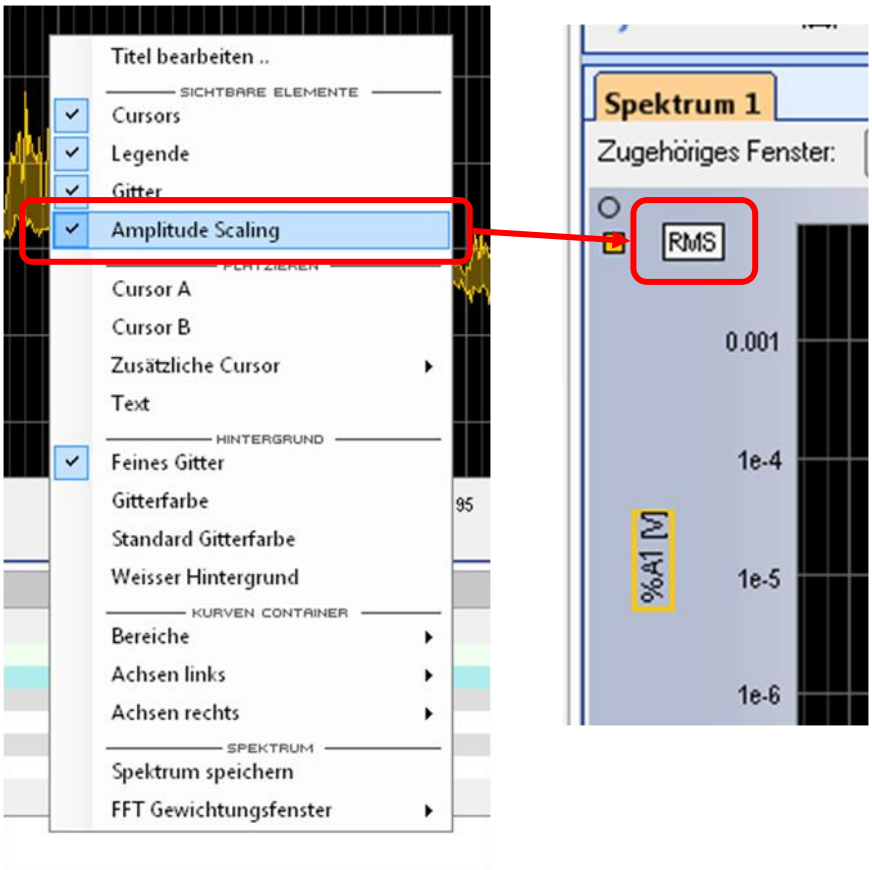
20 FFT Kurvenanzeige

20.1 Darstellung Skalierung



Bei einer FFT-Kurvenanzeige kann die Skalierung einer Y-Achse gewählt werden. Ein Rechtsklick auf der gewünschten Skala öffnet das entsprechende Popup-Menü.
Entweder wird die Achse **Linear**, **Logarithmisch oder als dB Unit** dargestellt. Bei "Phase" erfolgt das Anzeigen des Phasenverlaufs.
Die Skalierung kann als **Peak**, **RMS**, **Power** oder **RMS²** gewählt werden.

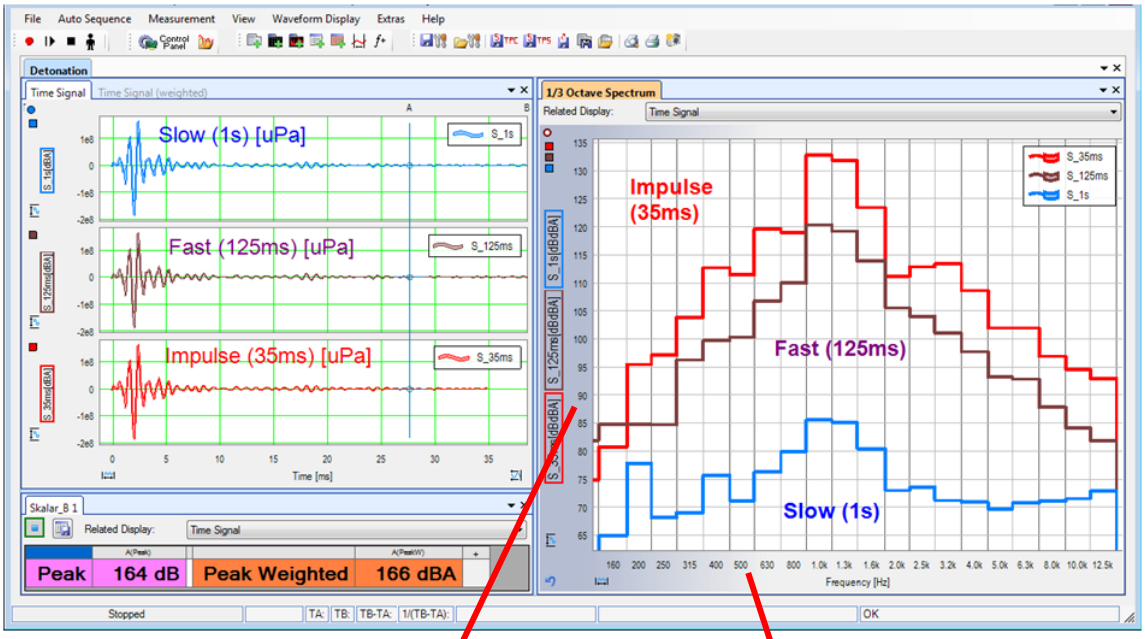
Um die eingestellte Skalierung oben auf den Y-Achsen anzuzeigen, kann durch einen Rechtsklick in die Kurvenanzeige die Option **"Amplitude Scaling"** eingeschalten werden.

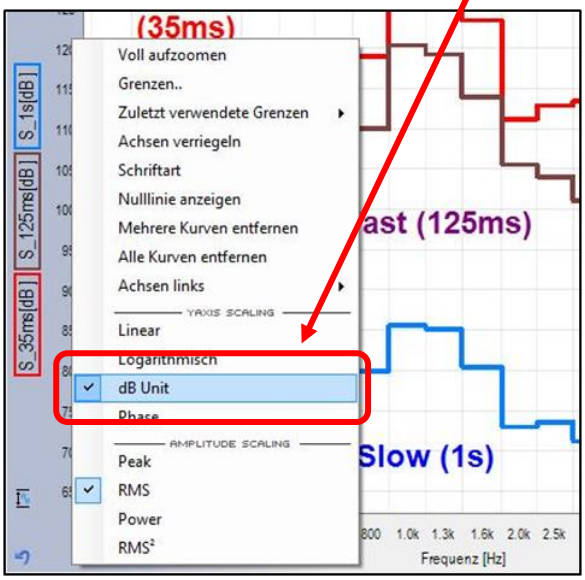


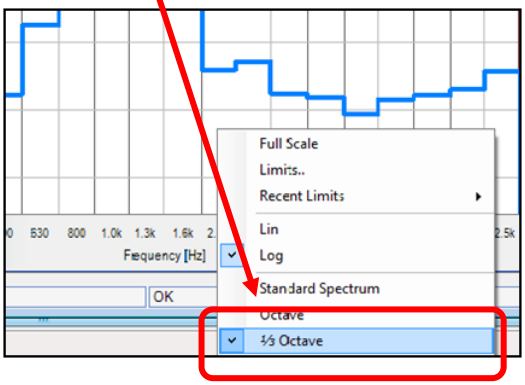
20.2 Oktaven- und Terz-Anzeigen

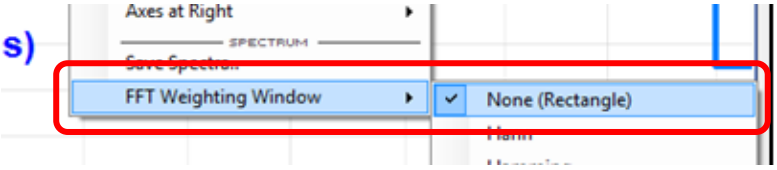
Das FFT Waveform Display wurde neben der bestehenden Darstellung als "Standard Spektrum" um die Optionen "Oktaven" und "⅓ Oktaven" (Terz) erweitert. Diese dienen vor allem für die Analyse bei akustischen Anwendungen.

Mit einem Rechtsklick auf der **Frequenz-Skala** kann im Pop-Up-Menu die "Octave" oder "1/3









Octave" (Terz) Darstellung gewählt werden. Für Octave- oder Terz-Darstellung muss die **Einheit auf der Y-Achse auf "dB Unit"** gestellt sein.

Normalerweise muss für Octave- oder Terz-Darstellung das **FFT-Gewichtungs-Fenster** ausgeschaltet werden (**"Rechteck"**).

21 Videos anzeigen, synchron zu Signal-Kurven

Video-Filme können in TranAX vor- und rückwärts abgespielt oder einfach als Standbild angezeigt werden. Filme lassen sich parallel zu aufgezeichneten Signal-Verläufen abspielen. Wenn nun diese Signale gleichzeitig mit den Videos aufgenommen wurden, lässt sich die Bildfolge synchron zu den Kurven-Verläufen analysieren.

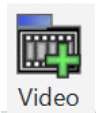
Die Filme können in gängigen Formaten aufgenommen worden sein. Am besten eignet sich das avi-Format. Vorbereitend sollten die Video-Dateien in ein Unterverzeichnis ***"videos"*** des TranAX-Experiments mit den zugehörigen Kurven-Dateien kopiert werden.

Datei Typ	Verzeichnis
Kurven-Dateien (*.tpc5)	...*experimentName.exp****data** Standard Verzeichnis der Kurven-Dateien
Filme (*.avi, *.mpg, usw.)	...*experimentName.exp****videos** Ev. dieses Verzeichnis vorab manuell erstellen

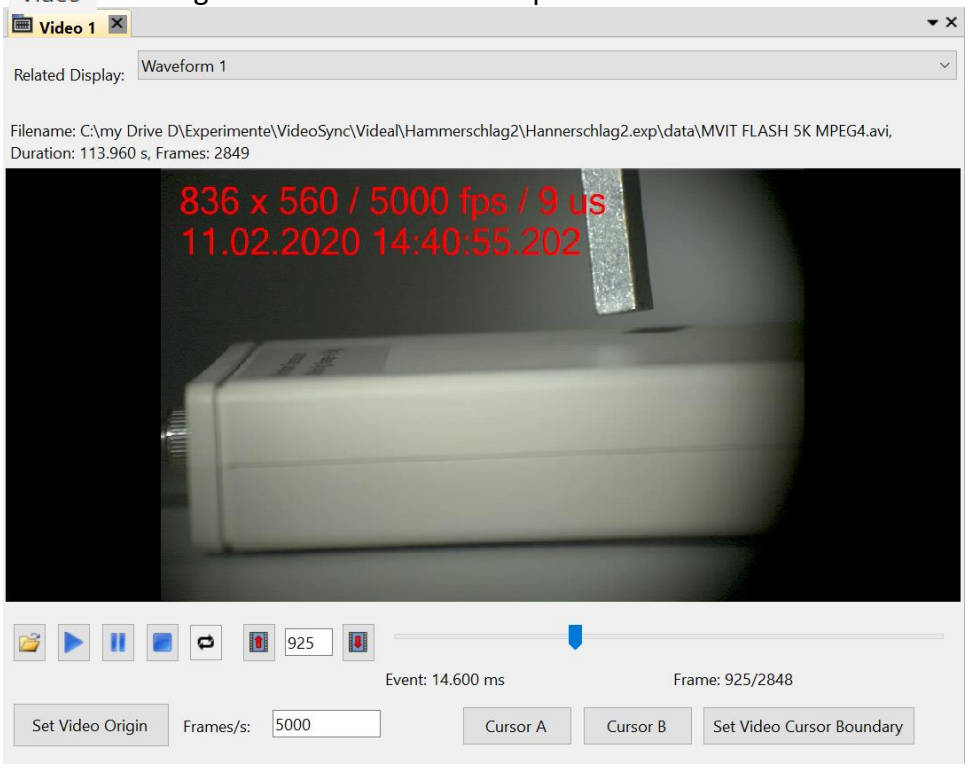
Ein Video besteht aus einem **Video-Container** (AVI, MP4, MOV, OGG, etc.) plus einem **Video-Codec** (h264, mpeg4, xvid, divx, etc.)

Seit **TranAX Version 4.1.2.2292** wurde die implementierte Video-Player-Engine, die auf dem VLC Media Player basierte, durch die **OpenCV-Engine** (www.openvc.org) ersetzt.
Gründe dafür waren eine bessere Unterstützung des verwendeten Dateiformats und eine flexiblere Verwendung für die Einzelbildvisualisierung.

Ein Video-Fenster wird via Ribbon Tab "Layout", "Documentation & Video", Klick auf das Icon

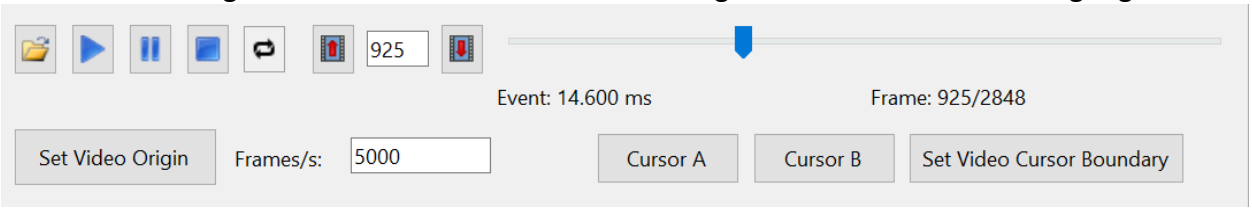


"Video" geöffnet und kann danach, falls erforderlich, wie gewohnt unplatziert werden. Beim Öffnen eines Video-Fensters wird auch gleich der Dialog zur Auswahl einer Video-Datei geöffnet. Ein in TranAX importierte Video Datei sieht wie folgt aus:



21.1 Die Verwendung des Video Waveforms

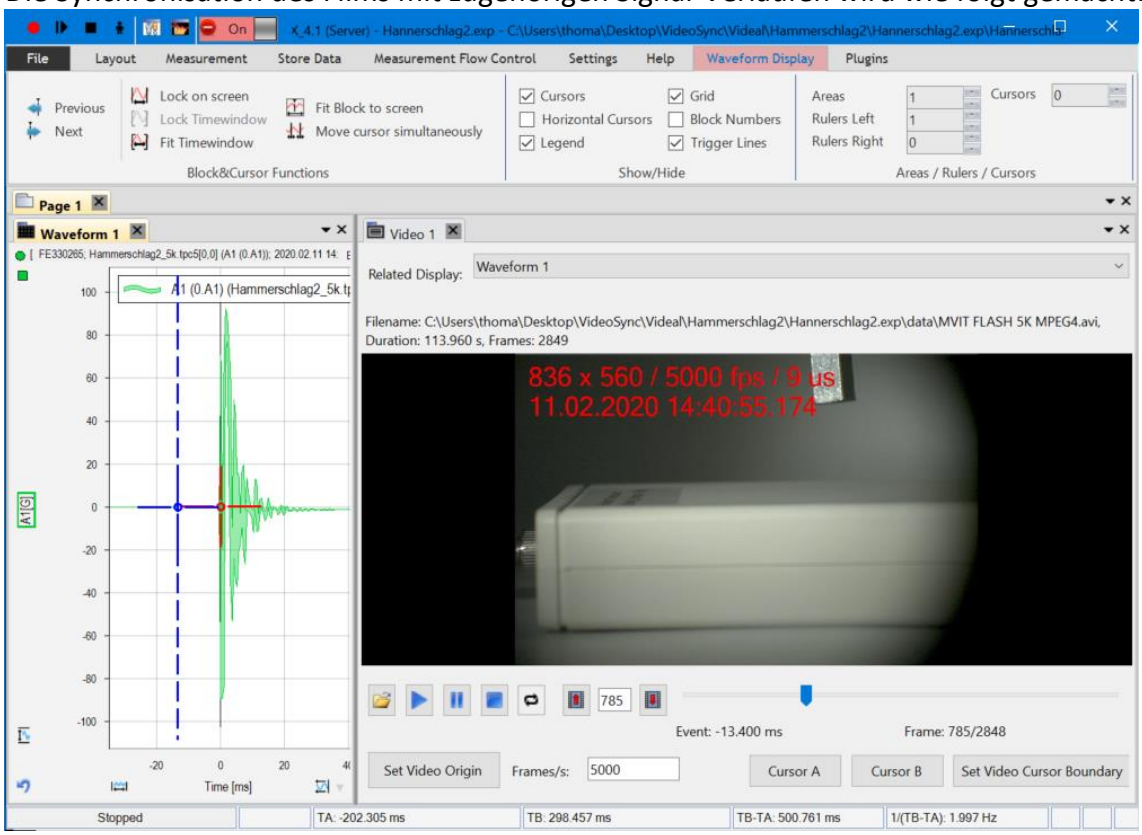
Für die Bedienung des Video Waveforms stehen die folgenden Elemente zur Verfügung:



Related Display: Waveform 1	Der Parameter "Zugehöriges Fenster" wird per Standard auf das aktivierte Hauptfenster mit Signal-Kurven gesetzt. Dieser kann auch im Nachhinein angepasst werden.
\data\MVIT FLASH 5K MPEG4.avi,	"Dateiname" zeigt die aktuell geöffnete Videodatei.
Duration: 113.960 s, Frames: 2849	"Länge" und "Frames" gibt eine Übersicht über die Spiellange und Anzahl der Frames im Video.
	Öffnet einen Dateidialog, um eine andere Videodatei auszuwählen. Bitte beachten Sie, dass die Einstellungen für die Bildrate erneut vorgenommen werden müssen.
	Startet die Videowiedergabe, Synchronisiert mit dem "Video Play" Marker im "Zugehörigen Fenster".
	Unterbricht die aktive Videowiedergabe.
	Stoppt die aktive Videowiedergabe und kehrt zum ersten Bild (Frame 0) zurück.
	Wiederholt die Videowiedergabe entweder während der gesamten Videodauer oder innerhalb der festgelegten Cursorgrenze.
	Geht ein Bild (Frame) zurück.
925	Aktuell dargestelltes Frame
	Geht ein Bild (Frame) vorwärts
	Mit dem Schieberegler für die Bildposition können Sie vom ersten zum letzten Bild wechseln. Wenn die Option "Limitiere Video auf Cursor" aktiviert ist, wird die Position des Schiebereglers durch die Cursor A und B auf die angegebene Position begrenzt.
Event: 14.600 ms	Zeigt den Zeitstempel der Position des "Video Play Markers" im Vergleich zum "Video Origin Marker" an.
Frame: 925/2848	Die aktuelle Frameposition, beginnend bei 0.
Set Video Origin	Setzt den Videoursprung im Bezug zum "Video Origin Marker" der YT-Waveform fest.

Frames/s: 5000	Die Framerate des aufgenommenen Videos. Muss definiert werden, damit das Video mit den Messkurven synchron ist.
Cursor A	Setzt Sie den Cursor A vom "Zugehörigen Fenster" auf die aktuelle Position des "Video Play Markers".
Cursor B	Setzt Sie den Cursor B vom "Zugehörigen Fenster" auf die aktuelle Position des "Video Play Markers".
Set Video Cursor Boundary	Spielt das Video innerhalb der definierten Grenze zwischen Cursor A und Cursor B ab und sperrt auch deren Position.

Die Synchronisation des Films mit zugehörigen Signal-Verläufen wird wie folgt gemacht:



- Zugehörige **Kurven-Signale** in das entsprechende Anzeigefenster **platzieren** (z.B. via Signalquellen-Browser). Ev. Zoomen und Verschieben der Kurve auf eine **markante Stelle im Signalverlauf**, wo auch eindeutig das entsprechende Bild im Film zugeordnet werden kann.
- An dieser markanten Stelle via **rechte Maus-Taste** in der Kurvenanzeige den **"Video Origin Marker"** setzen.
- Mit dem Parameter **"Frames/s" die Bildfrequenz** des Videos einstellen (dieser Wert kann nicht aus allen Video-Formaten übernommen werden, er sollte also in jedem Fall überprüft und richtig eingestellt werden).
- Den Film-starten und an der **markanten Stelle stoppen**.
- Danach mit Hilfe des Knopfes ***"Set Video Origin"*** das aktuelle **Bild** mit dem **Origin auf der Kurve synchronisieren**. Dabei wird der "Video Play Marker" auf das Video-Origin der Signal-Kurve gesetzt. Durch Verschieben des Video-Play-Markers kann das gewünschte Bild noch feiner angefahren werden. Ev. durch Aufzoomen an der entsprechenden Stelle den Kurven-Ausschnitt vergrössern. Danach erneut "Set Video Origin" anklicken.

Der Film kann nun vor- und rückwärts gespult werden. Dabei läuft der Video-Play-Marker im Kurven-Fenster mit. Wenn umgekehrt der Video-Play-Marker verstellt wird, läuft auch der Film entsprechend mit. Die Synchronisation sollte also gewährleistet sein.

Die korrekte Übereinstimmung ist natürlich nur gewährleistet, wenn bei ***"Frames/s"*** die richtige Bildfrequenz eingestellt ist.

22 Dokumentationsfenster

Das Dokumentation Fenster bietet die Möglichkeit, auf eine einfache und schnelle Weise eine Seite (A4, A3, etc.) mit Komponenten (Kurvenanzeigen, Skalartabelle, etc.) zu gestalten, welche dann gedruckt oder als PDF gespeichert werden kann.
Template Elemente können als Kopf- oder Fusszeile verwendet werden, um Textelemente, Bilder etc. zu platzieren, welche dann auf jeder Seite abgedruckt werden. In Textfelder kann man zudem auch Skalar-Resultate aus dem Formel-Editor abbilden.

Folgende Elemente sind für das Dokumentationsfenster vorhanden:

Kurvenanzeigen	
YT	YT
FFT	FFT
Zoom	Zoom
XY	XY
Marker	Marker

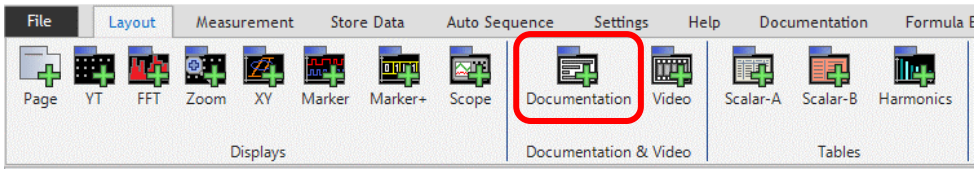
Tabellen	
Scalar-A	Skalar Tabelle A
Scalar-B	Skalar Tabelle B
Harmonics	Harmonische

Formen	
Line	Zeichnen von Linien
Rectangle	Zeichnen von Rechtecken

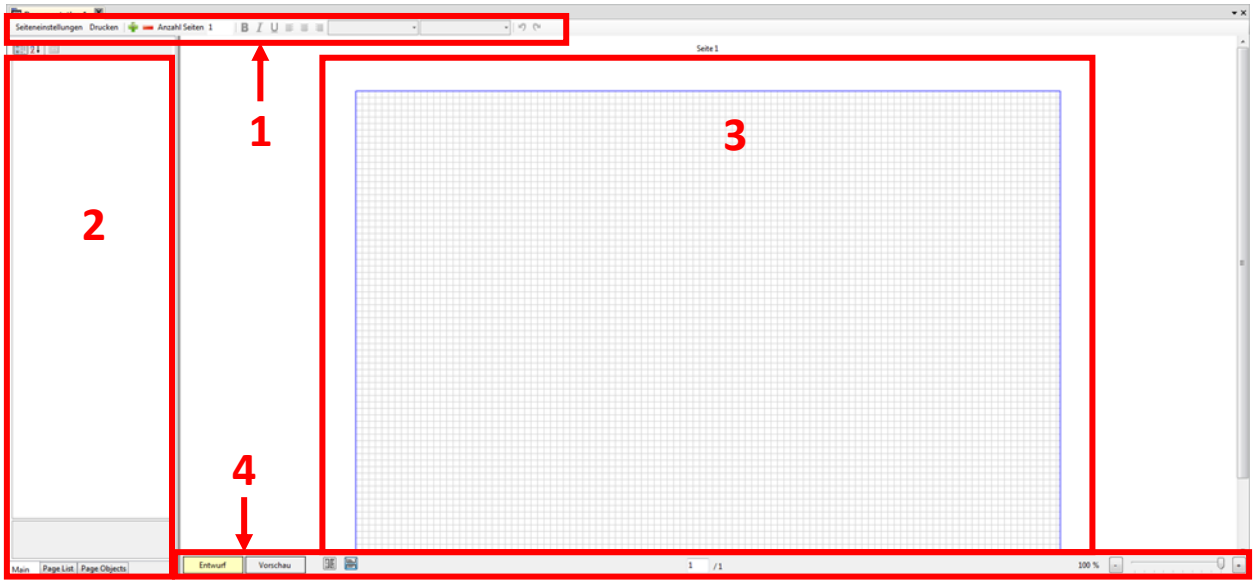
Diverses	
Text	Platzieren von Texten
Image	Platzieren von Bildern

Template	Templates (Verwendung für Kopf- und Fusszeilen)
RecLog	Aufnahmeprotokoll
Attributes	Attribute

Um ein Dokumentations-Fenster zu erstellen, geht man zum Ribbon Tab "Layout", Gruppe "Documentation & Video" und drückt auf "Dokumentation":



Danach erhält man eine leere Dokumentationsseite:



1. Menüleiste
In der Menüleiste kann man Seiteneinstellungen (A4, A3, Hochformat, Querformat, etc.) vornehmen, drucken, die Anzahl Seiten festlegen sowie die Schrifteinstellungen in einem Textfeld verändern.

2. Eigenschaftsfenster
Das Eigenschaftsfenster stellt jeweils die Eigenschaften des selektierten Elements in der Designfläche dar. Von Element zu Element können die Eigenschaften variieren.

3. Entwurfs- oder Vorschaufenster
Im Entwurfsfenster werden die einzelnen Elemente (YT, FFT, Skalartabelle, etc.) platziert. Man wählt in Dokumentations-Tab das gewünschte Element aus und zieht das Element mit gedrückter Maustaste auf bis es die gewünschte Grösse besitzt. Standardmässig ist dieses Fenster immer auf Entwurf eingestellt. Man kann jeweils in die Druckvorschau wechseln indem man in der Statusleiste auf Vorschau klickt.

4.Statusleiste

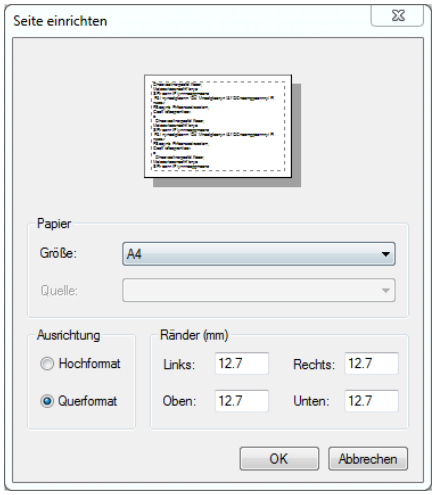
In der Statusleiste findet man die Möglichkeit, das Entwurfsmodus auf den Vorschaumodus umzustellen. Ebenfalls findet man hier die Möglichkeit, die Seitendarstellung entweder von links nach rechts oder von oben nach unten zu wählen.
In der Mitte befindet sich jeweils die Anzahl Seiten. Dort kann man jeweils auch gerade direkt auf die gewünschte Seite wechseln. Auf der rechten Seite findet man die Zoomeinstellungen (0-100%). Wenn die Zoomeinstellung unterhalb von 100% gewählt wird, werden die Elemente Skalartabelle A/B und Harmonische Tabelle nur noch gezeichnet dargestellt. Mit einem Doppelklick auf das jeweilige Element erscheint das Control in einem Fenster. Somit kann man dann das Element wie gewohnt konfigurieren.

22.1 Element platzieren

Ein Element (YT, FFT, Skalartabelle A/B, etc.) kann auf der Entwurfsoberfläche platziert werden. Dafür klickt man das gewünschte Element mit der linken Maustaste an und zieht mit gedrückter Maustaste das Element auf der Entwurfsoberfläche auf.

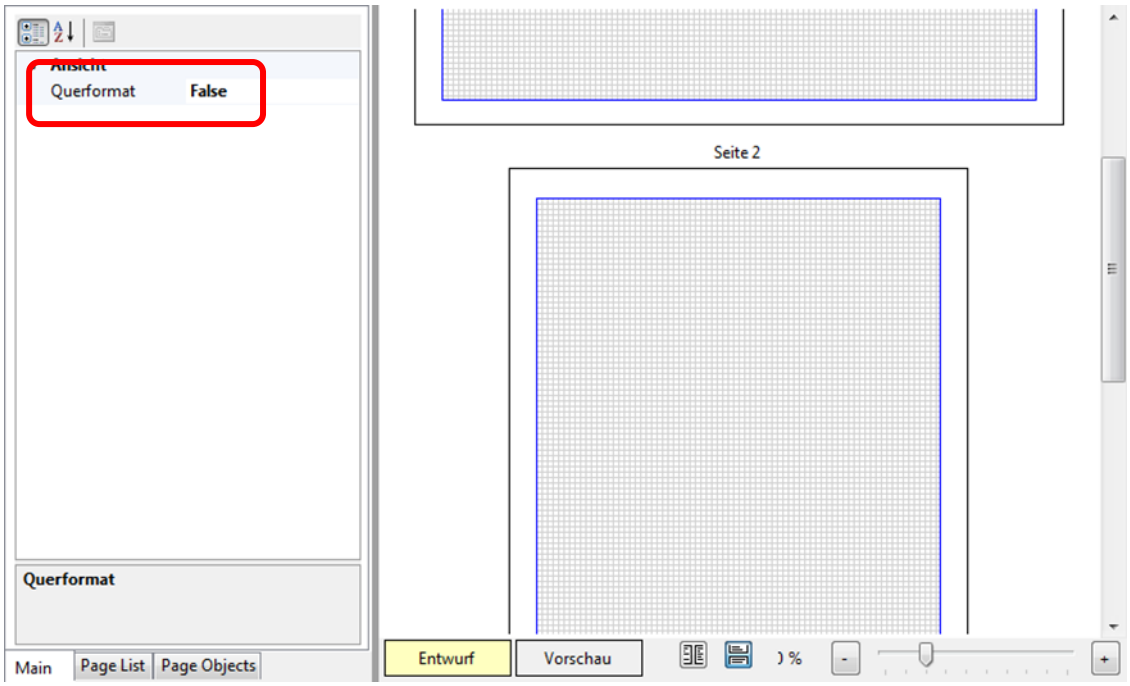
Stellen sie sicher, dass in der Statusleiste der Entwurf-Button angewählt ist bevor sie ein Element zu platzieren versuchen.

22.2 Seiteneinstellungen



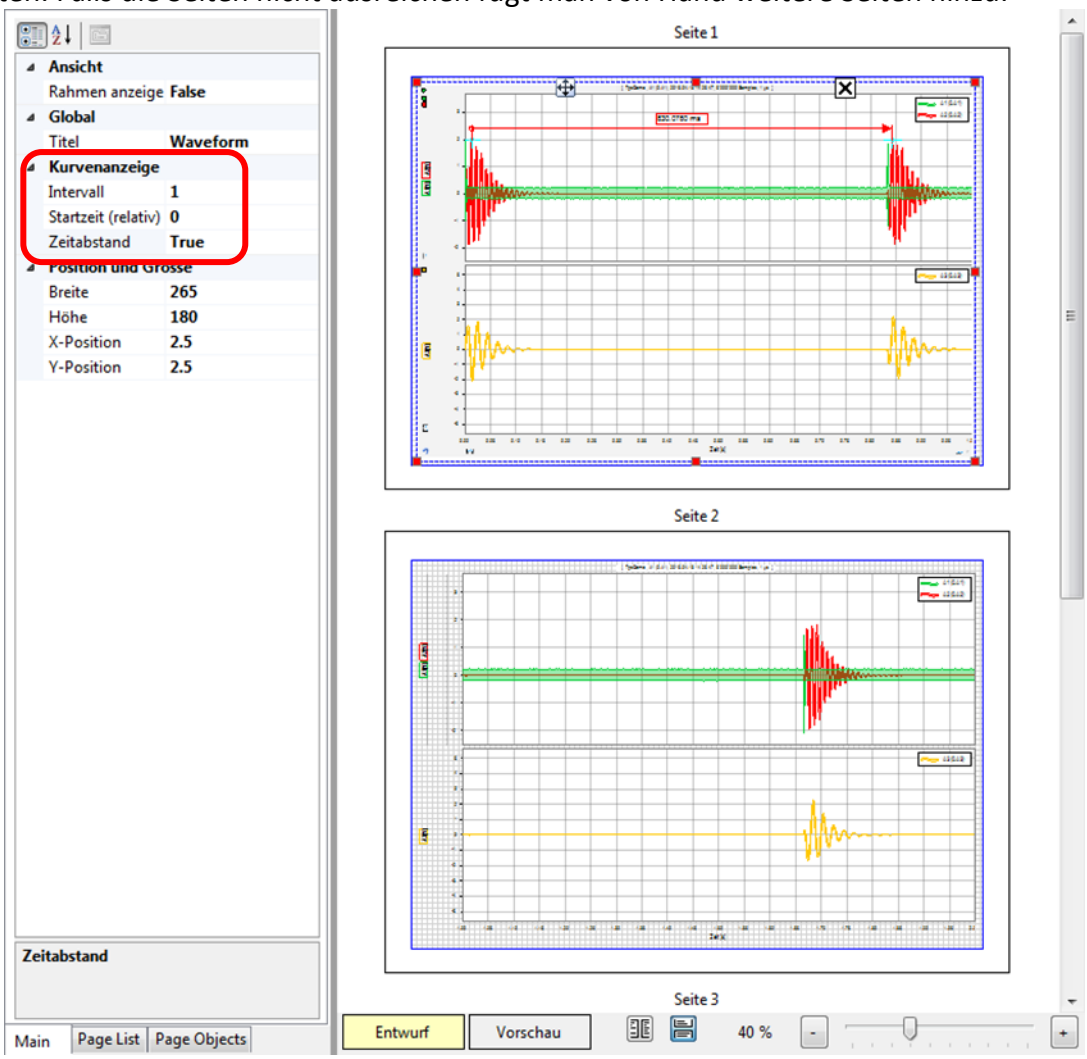
Um Seiteneinstellungen vornehmen zu können, drückt man in der Menüleiste auf **Seiteneinstellungen**. Danach erscheint ein gewohnter Windows Dialog:

Alle Einstellungen, die in diesem Dialog vorgenommen werden, werden dann auf alle Seiten in der Dokumentation angewendet. Es gibt jedoch die Möglichkeit, die Ausrichtung einzelner Seiten im Entwurfsfenster zu ändern. Dafür klickt man im Entwurfsfenster in einen leeren Bereich und stellt im Eigenschaftsfenster danach die gewünschte **Ausrichtung (Querformat)** ein:



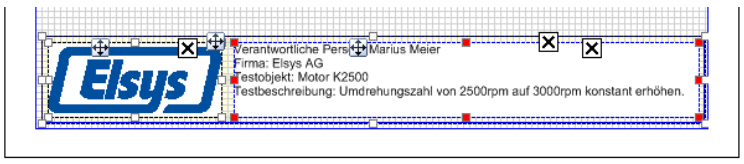
22.3 Kurvenanzeige auf mehrere Seiten aufteilen

Kurvenanzeigen können auf mehrere Seiten aufgeteilt werden. Dies könnte Beispielsweise erforderlich sein, wenn man eine Continuous-Aufnahme komplett dokumentieren möchte und den Detailgrad der Kurven möglichst hoch behalten will.
Um die Aufteilung vorzunehmen klickt man sein Kurvenfenster (Im Beispiel ein YT) an und stellt im Eigenschaftsfenster das **Intervall** (in Sekunden), die **Startzeit** (in Sekunden) ein und aktiviert den **Zeitabstand** (True). Danach verteilt sich die Kurvenanzeige automatisch auf die weiteren Seiten. Falls die Seiten nicht ausreichen fügt man von Hand weitere Seiten hinzu:

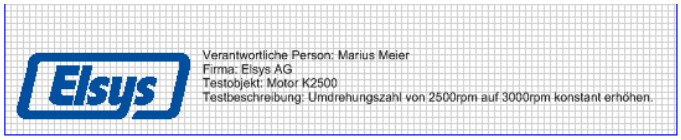


22.4 Templates (Kopf- und Fusszeile)

Templates können für Kopf- oder Fusszeilen verwendet werden. Ein Template kopiert sich selbst und den Inhalt jeweils auf die nächsten Seiten. Um ein Template zu erstellen wählen sie im Dokumentations-Tab das **Template Element** und setzten es auf Ihre gewünschte Seite.



In unserem Beispiel fügen wir ein Text-Element und ein Bild-Element in das Template ein. Wie man sehen kann, wird das Template danach auf alle darauffolgenden Seiten kopiert:



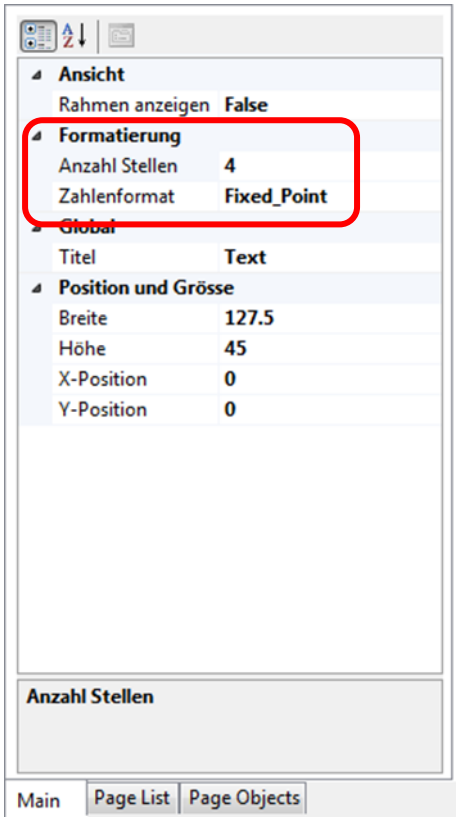
Falls man unterschiedliche Templates für Hoch- und Querformate verwenden möchte, kann man jeweils das gewünschte Template anwählen und im Eigenschaftsfenster die Einstellung **„Zeichne Template nur auf selben Blattformat“** anwählen.

22.5 Text – Spezielle Schlüsselwörter

Das Textfeld beinhaltet spezielle Schlüsselwörter, welche dann in der Druckvorschau sowie beim Ausdrucken mit den tatsächlichen Werten gezeichnet werden. Ein Schlüsselwort beginnt mit **"<%"** und endet mit **"%>"** In Folgender Tabelle finden sie alle Schlüsselwörter:

Schlüsselwort	Beschreibung
<%pages%>	Beinhaltet die Anzahl Seiten im Dokument
<%page%>	Beinhaltet die aktuelle Seite
<%date%>	Beinhaltet das aktuelle Datum

22.6 Text – Formel Editor

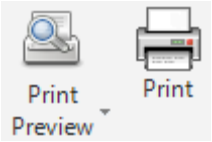


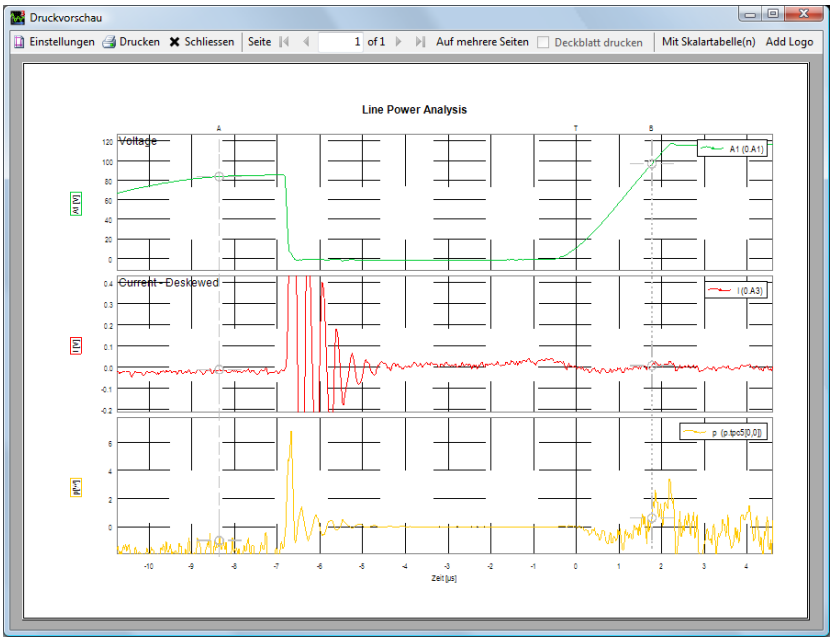
Um Variablen vom Formel Editor im Text-Element darzustellen, benötigt man lediglich den Variablennamen, welcher innerhalb von spitzen Klammern geschrieben werden muss (**<value>**). Zusätzlich kann über das Eigenschaftsfenster die Zahlenformatierung im Text-Element festgelegt werden:

22.7 Drucken (Legacy Mode)

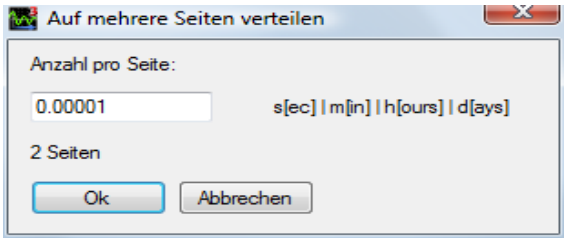
Der Druckdialog aus TranAX 3 sollte für neue Anwendungen nicht mehr verwendet werden, dieser ist aus kompatibilitätsgründen für bestehenden Anwendungen weiterhin verfügbar. Es wird empfohlen mit dem Dokumentation Fenster zu arbeiten.

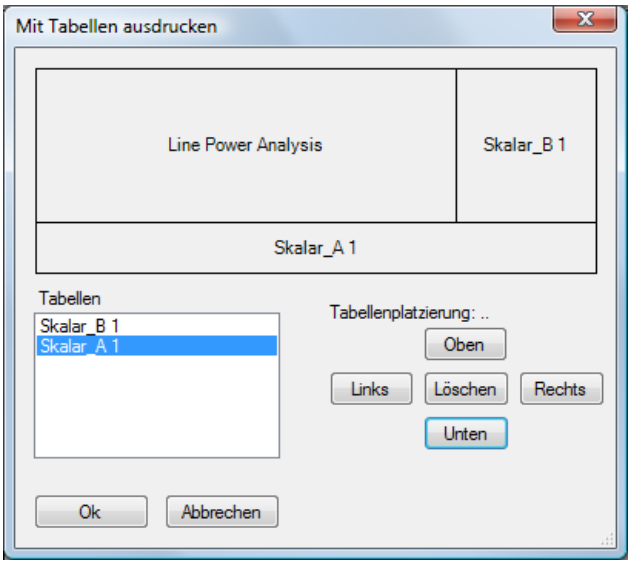
Um die aufgenommenen Kurven zu drucken wird zuerst das gewünschte **Kurvenfenster selektiert** und anschliessend die Schaltfläche "Drucken" im Ribbon Tab "Daten speichern" bei der Gruppe "Drucken & Snapshot" geklickt. Um zur Druckvorschau zu gelangen, kann im selben Tab die Schaltfläche "Druckeransicht" geklickt werden.



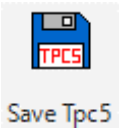


Bei der Druckeransicht öffnet sich ein Fenster, in dem eine Vorschau des Kurvenfensters angezeigt wird.
Je nach Monitor, Auflösung und Einstellung kann diese Ansicht von der Anzeige auf dem Monitor abweiche. Daher wird empfohlen mit dem Dokumentation Fenster zu arbeiten.

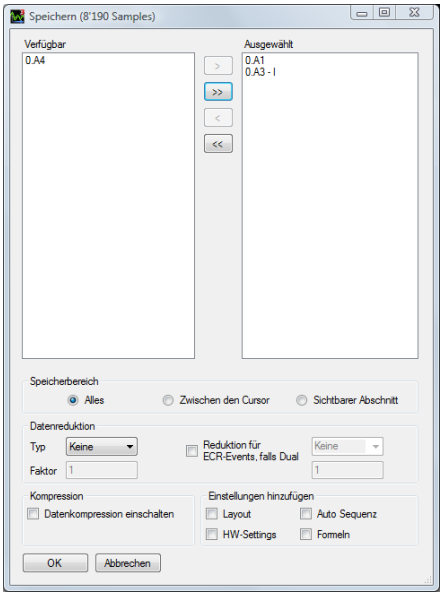
Einstellungen	Hier wird der Drucker, die **Papiergrösse** und die Orientierung gewählt.
Drucken	Dieser Knopf muss gedrückt werden, um die aktuelle **Kurvenanzeige** mit den allenfalls zugehörigen **Skalartabellen auszudrucken**.
Auf mehrere Seiten	Die **Zeitachse** kann zum Ausdrucken über **mehrere Seiten** gedehnt werden. Nachdem die Zeitspanne pro Seite eingegeben wurde, steht unterhalb der Eingabe auf wie viele Seiten der Ausdruck zu stehen kommt. 
Deckblatt drucken	Falls Attribute oder Fehlerprotokolle vorhanden sind, können diese Einträge an dieser Stelle in Tabellenform gedruckt werden.

Mit Skalartabelle(n)	Falls ein Kurvenfenster zugehörige Skalartabellen hat, muss definiert werden, an welcher Position diese Tabellen relativ zum Kurvenfenster ausgedruckt werden sollen. 
Add Logo	Dieser Knopf befindet sich oben rechts im Fenster Druckvorschau. Sollte ein Logo mit auf den Ausdruck, so wird das ausgehend von Add Logo hinzugefügt. Unterstützt werden die Formate bmp, jpg, gif und png.

23 Kurven speichern



Beim Ribbon Tab "Daten speichern kann mit Hilfe der Schaltfläche "TPC5 speichern" Signalkurven im HDF5-Format auf die Festplatte gespeichert werden. (Das HDF5-Format wurde definiert von NCSA, dem "National Center for Supercomputing Applications", USA).

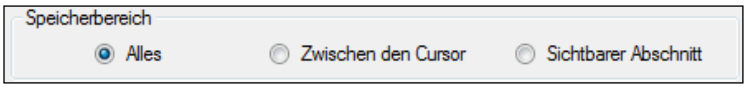


Die linke Liste zeigt die **verfügbaren Signale**, die zum Abspeichern ausgewählt werden können. Die **rechte Liste** zeigt die aktuell ausgewählten Kurven.
Zur Auswahl stehen alle Hardware-Kanäle des aktuellen Clusters sowie alle Kurven aus Dateien, welche sich in einem Kurvenanzeige-Fenster befinden.

Weitere Optionen zur Komprimierung finden Sie im Menü ***"Extras" / "Einstellungen" / "Import/Export"***.

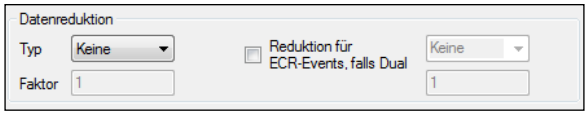
Wenn die gespeicherten Kurven später wieder in TranAX angezeigt werden sollen, müssen sie auf diese Weise im HDF5-Format (*.TPC5) abgespeichert werden.

23.1 Speicherbereich



Es kann ausgewählt werden, ob man die **gesamte Aufnahme**, nur den Bereich **zwischen den Cursor (A und B)** oder den aktuell **sichtbaren Bereich** abspeichern möchte.

23.2 Datenreduktion

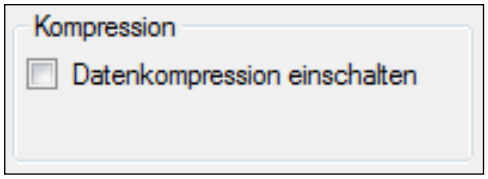


Die **Datenmenge** kann beim Abspeichern **verringert** werden. Es stehen verschiedene Varianten zur Verfügung:

- **Keine**: Keine Datenreduktion
- **Überspringen**: Es wird nur jedes n-te Sample abgespeichert.
- **Mittelung**: Für je n Samples wird der **Mittelwert** berechnet und dieser Wert wird abgespeichert.
- **MinMax**: Für je n Samples wird die kleinste und die grösste Amplitude gesucht und diese Werte werden abgespeichert. Bei dieser Variante werden für jede ausgewählte Kurve zwei Kurven abgespeichert, eine für die untere und eine für die obere Hüllkurvenbegrenzung.

Falls die Datenaufnahme im **ECR-Mode mit Dual** durchgeführt wurde, so kann für die schnellen ECR-Ereignisse und den Dual-Anteil eine unterschiedliche Datenreduktion gewählt werden.

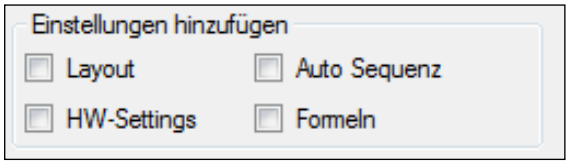
23.3 Kompression



Zusätzlich zur Datenreduktion können die Daten **komprimiert** werden. Das führt zu **10% bis 15%** kleineren Dateien, je nach Kurveninhalt.

Die **Kompression** ist **verlustfrei**, es gehen keine Daten verloren!
Je nach Menge der Daten kann das Speichern mehr Zeit in Anspruch nehmen.

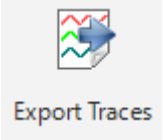
23.4 Einstellungen hinzufügen



Das TPC5-Format erlaubt nicht nur das Speichern von Messkurven sondern auch dasjenige des **Layouts**, der **HW-Einstellungen**, der **Autosequenz** und des **Formeln-Editors**.

23.5 Exportieren

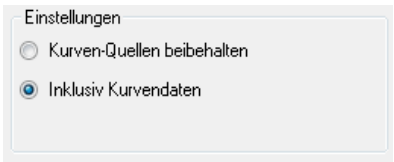
Im Ribbon Tab ***"Daten speichern" / "Exportieren"*** via ***"Kurven"*** können Signalkurven in den Formaten TPC (TransAS 2 - kompatibel), ASCII, DIAdem, Krenz usw. exportiert werden.



23.6 Speichern von Kurvenanzeige-Seiten

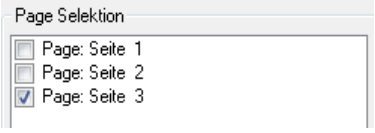
Mit der Aktion **"Seite speichern"** kann eine Untermenge von **allen Layout-Einstellungen** inklusiv Messdaten verwaltet werden. Dabei werden alle Anzeige-Fenster und Skalar-Tabellen in der entsprechenden Seite (mit dem Fokus) in eine Datei gespeichert.
Falls in der Maske dazu unter Einstellungen **"Inklusive Kurvendaten"** gewählt ist, werden von allen in den Anzeigefenstern enthaltenen Kurven auch deren Kurvendaten mitgespeichert. Dabei werden auch die Quellen-Angaben der Kurven in den Fenstern angepasst, sodass nach dem Laden einer gespeicherten Seite auch auf die Kurven in der Datei gezeigt wird. Dieses Feature ist vor allem nützlich, wenn bei einer **späteren Auswertung** das **Erscheinungsbild identisch** sein soll wie bei der Aufnahme, wo die Kurvendaten in der Regel unmittelbar von den Hardware-Kanälen herkommen. Bei der Auswertung wären die Kurvendaten nicht mehr unter diesen Quellenangaben vorhanden.

Das Speichern von Seiten kann über das Ribbon Tab ***"Daten speichern"*** in der Gruppe ***"Daten speichern und laden"***, bei der Schaltfläche ***"Seite speichern"*** vorgenommen werden. Danach öffnet sich der Dialog, um die Seite(n) zu speichern. Nun gibt man den gewünschten Dateinamen ein. Auf der rechten Seite hat man globale Einstellungsmöglichkeiten:



Kurven-Quellen beibehalten:
Es werden lediglich die **Referenzen** (Quellen-Angaben) auf die Kurven in die Seiten-Datei gespeichert.

Inklusiv Kurvendaten:
Es werden auch **alle Kurven** in die Datei abgespeichert und die Referenzen entsprechend angepasst.



Hier kann man auswählen, welche Seiten in die Datei gespeichert werden sollen. Beim Öffnen des Dialogs ist jeweils nur die Seite mit dem Fokus angewählt.

Die Datei wird mit der Endung ***.tdp** in das Data Verzeichnis des aktuellen Experiments abgelegt.

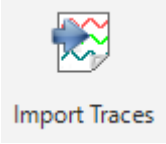
Über das Ribbon Tab ***"Daten speichern"***, in der Gruppe ***"Daten speichern und laden"***, bei der Schaltfläche ***"Seite laden"*** können gespeicherte Seiten wieder geladen werden. Im Dialog wählt man die entsprechende Datei und die daraus zu ladenden Seiten.

Die Seiten-Datei (*.tdp) kann ebenfalls im Signalquellen-Browser geöffnet werden. Daraus kann dann auf die gespeicherten Kurven zugegriffen werden.

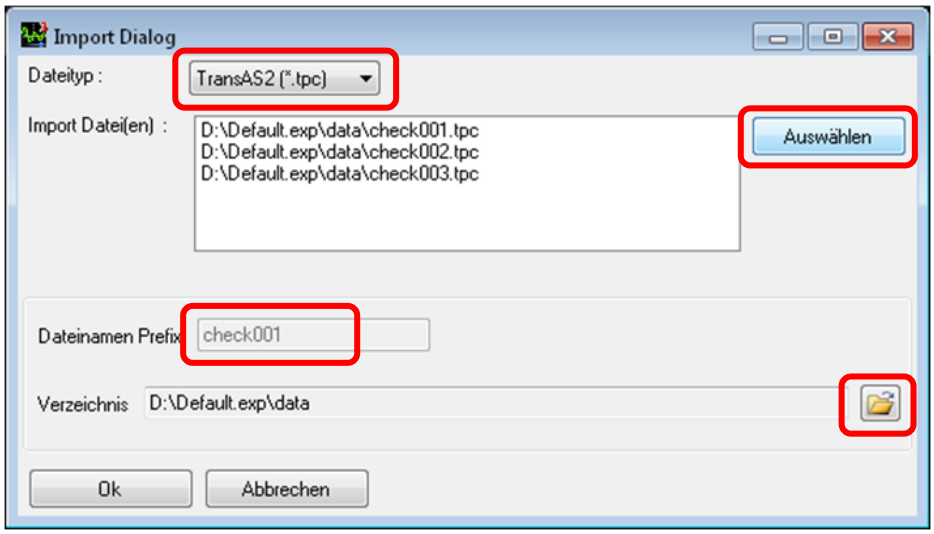
23.7 Import-Option

Verschiedene Datei-Formate lassen sich in TranAX importieren. Dabei werden diese Dateien ins TPC5-Format konvertiert (kopiert), sodass sie danach in TranAX analysiert werden können. Folgende Formate können importiert werden:

- TPC-Dateien (aus TranAX-Version 2)
- ASCII-Dateien (ASD-Dateien, auch aus TranAX-Version 2)
- Allgemeine ASCII-Dateien (Importieren via Wizard)
- Audio-Dateien



Über das Ribbon Tab ***"Daten speichern"***, Gruppe ***"Importieren"*** bei der Schaltfläche ***"Kurven"*** kann die Import-Maske geöffnet werden. Falls dieser Menü-Eintrag nicht vorhanden ist, muss die Option zuerst aktiviert werden.

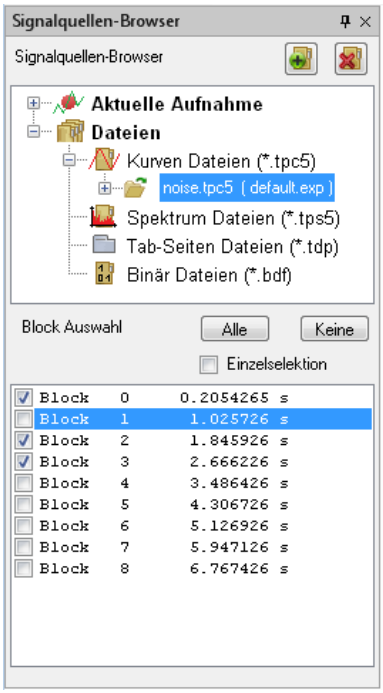


Mit dem Parameter ***"Dateityp"*** kann das Datei-Format (TPC, ASCII, Audio etc.) gewählt werden. Danach muss via ***"Auswählen"*** eine oder mehrere Original-Dateien selektiert werden. Eventuell muss man dabei zuerst in das Verzeichnis der Original-Dateien navigieren.
Falls nur eine einzige Datei gewählt wurde, kann mit dem Parameter ***"Dateinamen .."*** ein (ev. neuer) Name gewählt werden. Ansonsten werden die Namen der Original-Dateien übernommen. Die Namenserweiterung "TPC5" wird automatisch angehängt.
Unter ***"Verzeichnis"*** muss der Zielort der erzeugten TPC5-Dateien eingestellt werden. Dies ist in der Regel der "data"-Ordner des aktuellen Experiments.
Eine bereits existierende TPC5-Datei wird ohne Warnung überschrieben.

Die konvertierten TPC5-Dateien werden automatisch im Signalquellen Browser hinzugefügt. Dieser lässt sich via Menüleiste  öffnen. Via Signalquellen-Browser können ja dann die Kurven in die Anzeige-Fenster gezogen werden.

24 Signalquellen-Browser

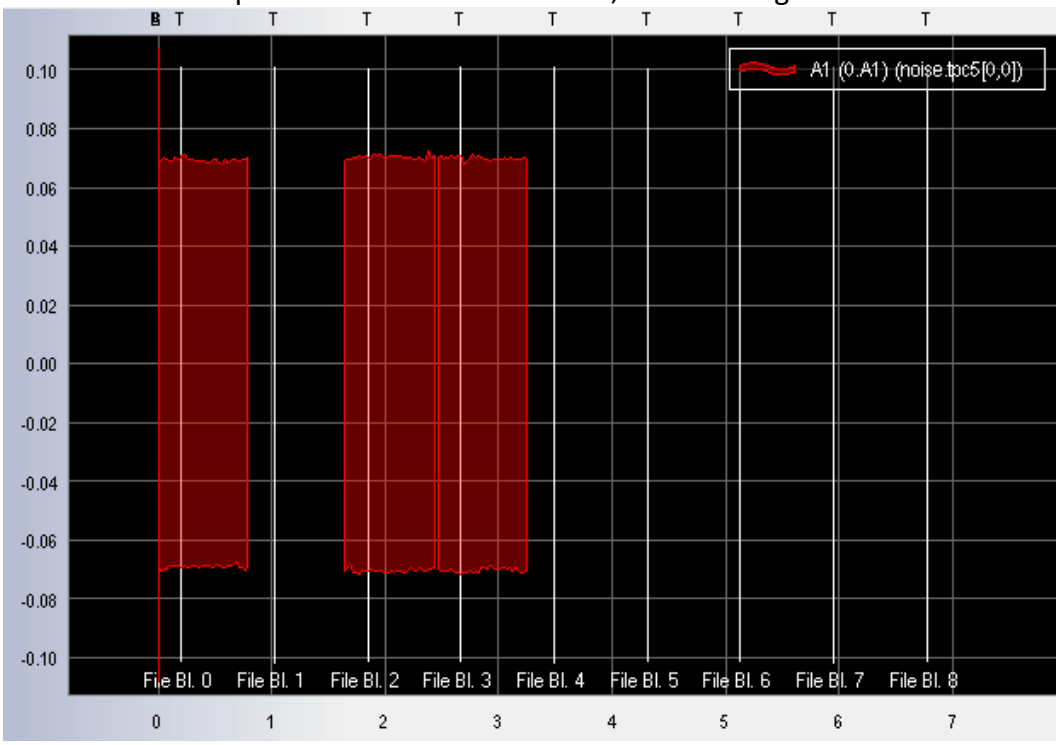
Der Signalquellen-Browser kann in der Menü Leiste mit der Schaltfläche geöffnet werden. Dieser wird in der Regel am linken Bildrand platziert.



Zusätzlich zum Control Panel werden auch alle Kanäle im **Signalquellen-Browser** unter "Aktuelle Aufnahme" aufgelistet. Mittels Drag & Drop werden die Daten als Signalkurven in der Kurvenanzeige dargestellt.
Um auf gespeicherte Kurven zugreifen zu können, kann via die Schaltfläche (oben rechts im Browser-Fenster) auf die gewünschte TPC5-, TPS5- oder TPD -Datei navigiert werden.

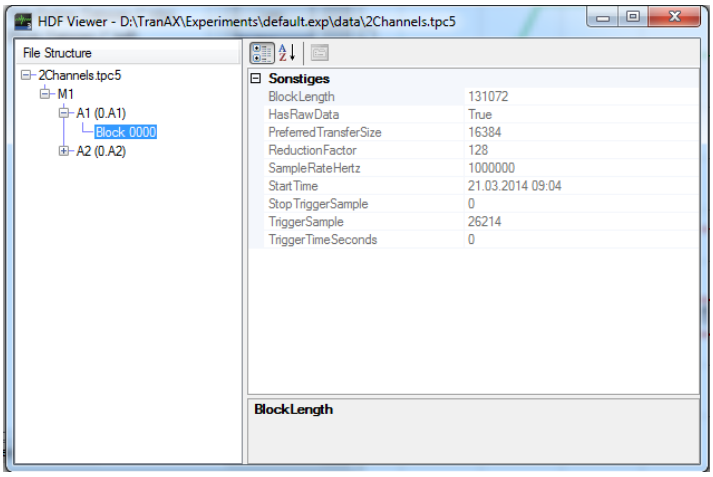
Für Aufnahmen im Multiblock- oder ECR-Mode zeigt der untere Bereich des Fensters jeden aufgezeichneten Block zum entsprechenden ausgewählten Kanal. Mit der Schaltfläche **"Keine"** werden alle Blöcke deaktiviert, die **"Alle"**-Schaltfläche zeigt wieder alle Blöcke an.

Mit einem Klick auf die entsprechende Schaltfläche werden Blöcke entweder hinzugefügt oder entfernt. Die ausgewählten Blöcke mit dem Häklein werden dann in der Kurvenanzeige dargestellt. Hier im Beispiel werden die drei Blöcke 0, 2 und 3 dargestellt.



24.1 HDF Viewer

Mit dem HDF Viewer kann man eine HDF-Datei (*.TPC5, *.TPS5, *.TDP) betrachten. Man erhält alle dazugehörigen Metainformationen, welche in die Datei gespeichert wurden. Mit einem Rechtsklick auf eine HDF-Datei wählt man im Kontextmenu "HDF Viewer öffnen".



24.2 Excel-Importer

Wenn man beim Setup den **Excel-Importer** installiert hat, erhält man im Kontextmenu einer HDF5 Datei den Eintrag "Nach Excel exportieren...". Somit hat man die Möglichkeit direkt aus dem Signalquellen Browser eine oder mehrere Kurven in eine Excel-Mappe zu übertragen.

25 Skalar Funktionen

TranAX gibt direkten Zugriff auf **über 60 eingebaute Skalar Funktionen**. Eine Skalar Funktion ist eine Messung oder **Berechnung** von **Kurven-Parametern** wie Maximum, Spitzenwert, Effektivwert oder der Phase zwischen zwei Kurven.

Eine komplette Liste der Skalar Funktionen ist im Anhang gelistet.

Auf den Kurven können Markierungen angezeigt werden, die sich auf entsprechende Resultate bei Skalar-Berechnungen beziehen. Es sind dies kleine Kreise bei Amplituden-Werten (z.B. Maximum, Peak-Peak usw.) oder kleine Rechtecke bei Zeit-Angaben (z.B. RMS-Periodic usw.). Dies erfordert eine globale Einstellung unter "Extras / Einstellungen / Benutzeroberfläche / Markierungen auf Kurve anzeigen".

Die Skalar Funktionen können in zwei verschiedenen Tabellen aufgesetzt und konfiguriert werden. Das erlaubt das Ausmessen von mehreren Kurven aus der letzten Datenerfassung oder abgespeicherten Kurven aus Dateien.

Die Skalar Funktionen Tabelle A ist eine matrixförmige Struktur von Kurven gegenüber Skalar Funktionen. Sie ist sehr schnell aufgebaut und ideal für Anwendungen mit **vielen Kurven,** aus denen **die gleiche Funktion** berechnet werden soll.

Skalar_A 1

Zugehöriges Fenster: Kurvenanzeige 1

Kurve	A	B	B-A	RMS	PP	Freq
A1 (0.A1) [A,B]	-1.263 V	-0.029 V	1.234 V	0.672 V	3.519 V	12.736 kHz
A2 (0.A2) [A,B]	0.315 V	-0.575 V	-0.889 V	0.802 V	3.150 V	19.929 kHz
A3 (0.A3) [A,B]	0.403 V	-0.072 V	-0.475 V	0.589 V	2.469 V	9.003 kHz
A4 (0.A4) [A,B]	1.475 V	-0.168 V	-1.643 V	0.620 V	2.413 V	4.286 kHz

Die Skalar Funktionen Tabelle B auf der anderen Seite ist **sehr flexibel** und wurde entwickelt für **komplexe Skalar Funktionen**. Diese Art von Tabelle gibt die Möglichkeit, per Funktion eine Primärkurve, falls benötigt eine Referenzkurve, das Cursor-Paar sowie die Messschwelle mit Hysteresis individuell festzulegen.

Skalar_B 1

Zugehöriges Kurvenanzeigefenster: Line Power Analysis

Papp(U,I)	Preal(U,I)	Preact(U,I)	PF(U,I)	Phase(U,I)	+
47.6409 VA	30.74 W	36.40 var	0.65	-14.75 °	

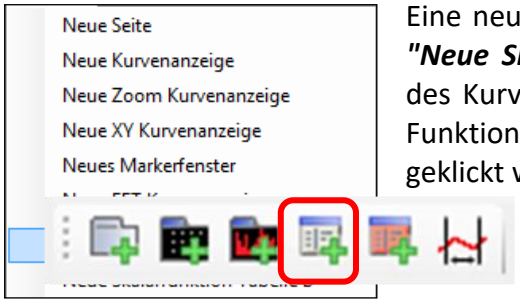
Die Kapitel Skalar Funktionen Tabelle A und Skalar Funktionen Tabelle B beschreiben Schritt für Schritt, wie eine neue Skalar Funktion Tabelle mit den gewünschten Kurven geöffnet und konfiguriert wird.
Die Skalar Funktionen sind im TranAX Skalar Funktions Dialogfenster und im Kapitel Beschreibung der Skalar Funktionen dieses Handbuches dokumentiert.

Eine neue Skalar Funktions Tabelle wird entsprechend der Grundeinstellung platziert, die im Menü ***"Extras" / "Einstellungen" / "Benutzeroberfläche" / "Standard Tab-Fenster-Platzierung"*** definiert ist.

Die Werte der markierten Skalartabellen können mit **copy & paste** (<Ctrl>+c / <Ctrl>+v) in eine andere Anwendung (z.B. Excel) übertragen werden.

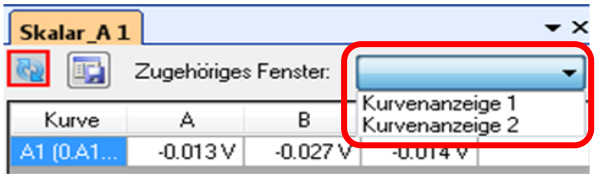
Mit **<Shift> + <Ctrl>+c** wird bei der **Skalartabelle A** auch die **Titellinie** mit übertragen.

25.1 Skalar Funktionen Tabelle A



Eine neue Skalar Funktions Tabelle kann mit Menü ***"Ansicht" / "Neue Skalar Funktion Tabelle A"*** geöffnet werden. Innerhalb des Kurvenfensters wird ein neues Registerblatt mit der Skalar Funktion-Tabelle kreiert. Alternativ kann auch das Icon im Menü geklickt werden.

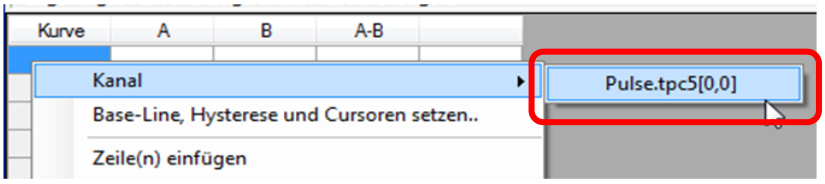
Standardmässig werden die **Amplitudenwerte von Cursor A, Cursor B** und deren **Differenz** in die Tabelle eingefügt. Das zugehörige Kurvenfenster, also die **Quelle** der zur Verfügung stehenden Kurven, kann im Dialog **"Zugehöriges Fenster"** ausgewählt werden. Hier werden alle verfügbaren Kurvenanzeigen aufgelistet. Die neu erstellte Tabelle kann nun sehr **flexibel positioniert** werden.

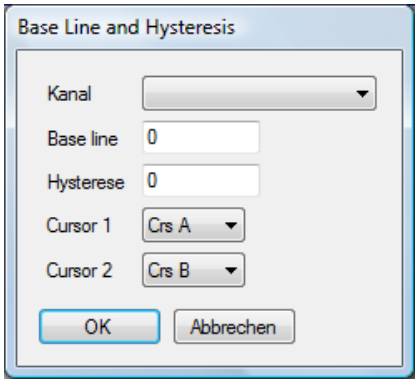


Durch Ziehen des Registerblattes ***"Skalar_A 1"*** lässt sich die Tabelle auf einer beliebigen Seite des aktiven Kurvenfensters platzieren. Natürlich kann die Tabelle auch als eigenes Registerblatt verwendet werden.

25.1.1 Kanalauswahl zur Berechnung der Skalar Funktionen

Per **Rechtsklick** auf die erste freie Zelle unterhalb des Spaltenkopfes ***"Kurve"*** kann eine an wählbare Kurve selektiert werden.



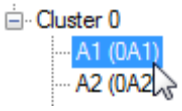


Alternativ dazu kann mit einem Doppelklick in der Kurvenspalte das Fenster **Base Line and Hysteresis** geöffnet werden. In diesem Dialogfenster können der Kanal, die Basislinie für allfällige Nulldurchgänge, die Hysterese dazu und das Cursor-Paar, das als Messfenster dient, festgelegt werden.

Sobald eine Kurve selektiert wurde, werden die Cursor-Werte ermittelt und in der Tabelle dargestellt.

Die Kurven, die in der **Skalar Funktions Tabelle** per Zeile ausgewählt werden können, sind im zugehörigen **Kurvenfenster** vorhanden.

Kurven können auf zwei verschiedene Arten mittels Drag & Drop in die Spalte *"Kurve"* gezogen werden:

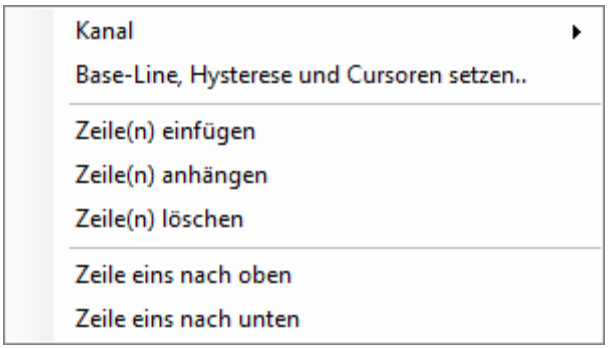


Vom Signalquellen Browser

vom Kanal-Schalter im Kurvenfenster

Die Skalar Funktion Tabelle wird normalerweise ausschliesslich im **Single Trigger-Mode** oder in einer Schleife einer Autosequenz mit entsprechender Wartezeit aktualisiert. Falls Auto-Refresh aktiviert ist, so wird die Tabelle auch dann aktualisiert, wenn Single Trigger Mode ausgeschaltet ist.

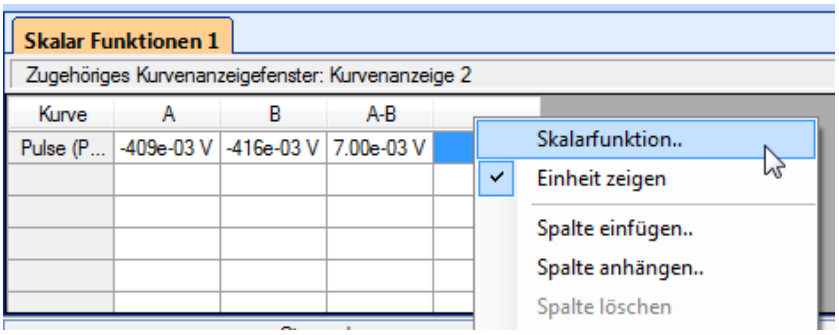
Die Basislinie und die Hysteresis müssen bei Bezugspegel-basierenden Skalaren gesetzt werden.



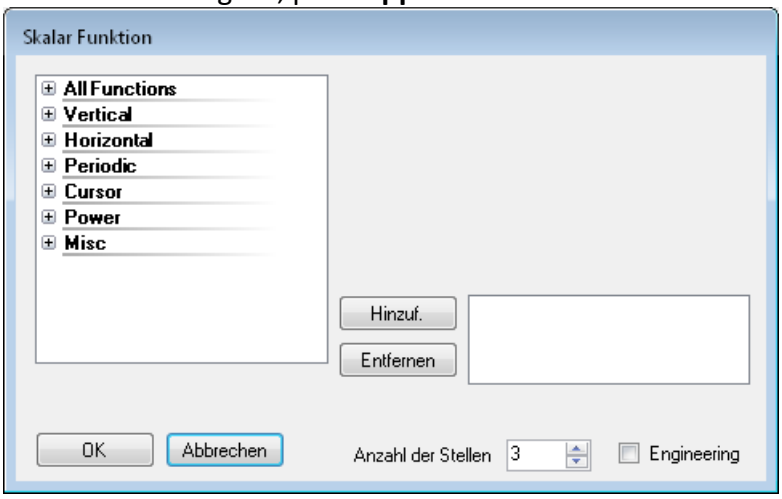
Im **Kurven-Kontextmenü** (rechte Maustaste) können **neue Zeilen** eingefügt, gelöscht oder angehängt werden. Auch die Reihenfolge der Zeilen lässt sich ändern mit *Zeile eins nach oben* und *Zeile eins nach unten*.

25.1.2 Auswahl der Skalar Funktion

Eine Skalar Funktion kann auf verschiedene Arten einer Kolonne zugewiesen werden. Eine Möglichkeit ist via Rechtsklick auf die Kopfzeile einer Kolonne das Kontextmenü öffnen und *Skalar Funktion..* auswählen.



Es ist ebenso möglich, per **Doppelklick** direkt ins Fenster **"Skalar Funktion"** zu gelangen.



Die **Skalar Funktionen** sind in verschiedene **Kategorien** unterteilt:

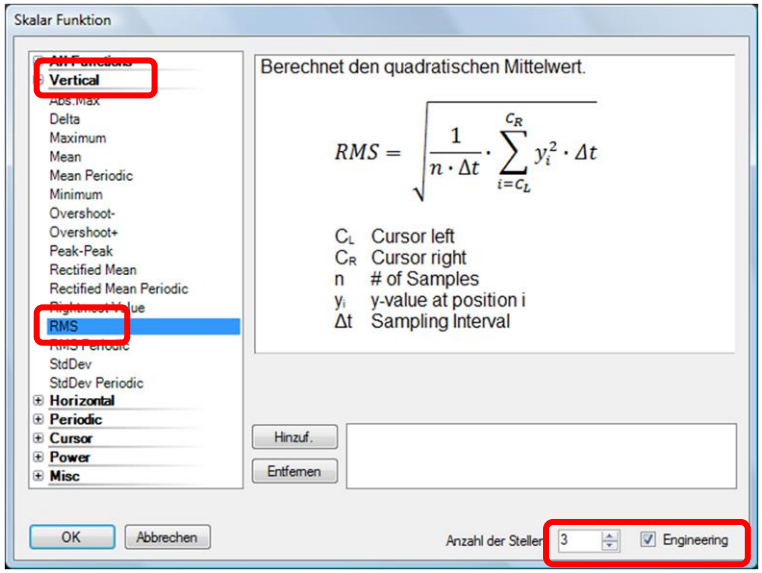
- **All Functions:** Listet alle Skalar Funktionen auf.
- **Vertical:** Listet die Skalar Funktionen auf, die vertikale Parameter der Kurve messen.
- **Horizontal:** Listet die Skalar Funktionen, die horizontale Parameter der Kurve berechnen. Diese Skalare sind typischerweise Zeitmessungen und benötigen meistens als Zusatzinformation die Basislinie und Hysterese.
- **Periodic:** Listet die periodischen Skalar Funktionen auf. Diese Skalare können nur auf periodische Signale angewendet werden. Sie benötigen als Zusatzinformation die Basislinie und Hysterese für die Ermittlung der ganzen Perioden.
- **Cursor:** Listet die Skalar Funktionen auf, die sich auf Cursor-Messungen beziehen.
- **Power:** Listet die Skalar Funktionen auf, die sich auf elektrische Leistungsmessungen beziehen.
- **Misc:** Listet spezielle Skalar Funktionen auf wie "Area" oder "Number of Triggers", die keiner der oben genannten Kategorien angehören.

Einige Skalar Funktionen werden bezogen auf eine bestimmte Basislinie mit Hysterese ausgewertet. Im Kapitel Beschreibung der Skalar Funktionen sind die Informationen betreffend Abhängigkeit der Basislinie angegeben.

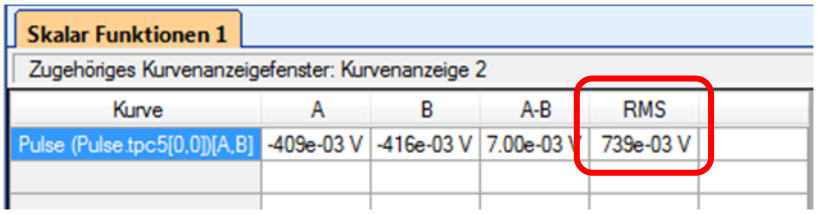
25.1.3 Beispiel 1: RMS

- *RMS* - Vertikale Skalar Funktion; ohne Referenzkanal; ohne Basislinie und Hysterese

Doppelklicken Sie auf die **Kopfzeile** einer Kolonne, um die aktuelle Skalar Funktion zu ersetzen. Klicken Sie im Dialog "Skalar Funktion" auf die **Funktions-Gruppe "Vertical".**
Selektieren Sie die **Funktion "RMS"**. Die Berechnungs-Formel mit Beschreibung und Legende wird im Fenster rechts der Liste angezeigt. Zur Darstellung des Messwertes wird **Engineering Mode** angewählt mit **drei Nachkommastellen**. TranAX merkt sich diese Einstellung und wird diese als Standard für die nächsten Funktionen verwenden.
Klicken Sie anschliessend "OK" um die Einstellungen zu akzeptieren und die Funktion "RMS" in die Tabelle einzufügen. Die vorherige Skalar Funktion wird ersetzt.



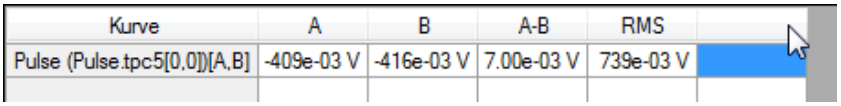
Der **RMS-Wert** wird aus der angewählten Kurve zwischen den **Cursor A und B** berechnet und in der **Spalte "RMS" angezeigt**. Rechts der Spalte "RMS" wurde automatisch eine **neue Spalte angehängt**, um das Hinzufügen einer weiteren Funktion zu erleichtern.



Normalerweise werden die **Skalar-Berechnungen** zwischen den selektierten **Cursors durchgeführt** (siehe Beschreibung der Skalar Funktionen). Unter Kanalauswahl wurde beschrieben, wie die Cursors ausgewählt werden. Im zugehörigen Kurvenfenster lassen sich die Cursors visualisieren und platzieren, womit das Messfenster pro Zeile definiert wird. Durch **unterschiedliche Cursor-Paare** kann das Messfenster pro **Zeile individuell definiert** werden.

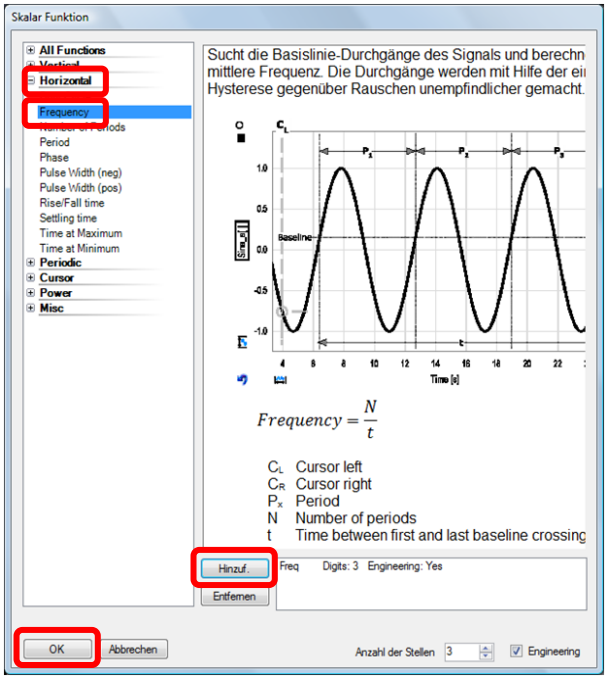
25.1.4 Beispiel 2: Frequency

- *Frequency* - Horizontale Skalar Funktion; ohne Referenzkanal; mit Basisline und Hysterese



Doppelklick auf die freie Kolonnen-Kopfzeile, um das Fenster **Skalar Funktion** zu öffnen.

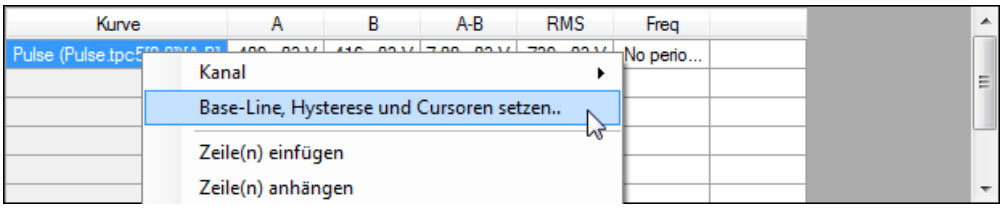
Anschliessend öffnen Sie die Liste **"Horizontal"** und selektieren die **Funktion "Frequency"**. Die Berechnungsformel und eine Grafik werden rechts der Liste angezeigt. Der Anzeige-Mode ist nach wie vor Engineering mit drei Nachkommastellen.
Klicken sie den Knopf "Hinzuf.", um den Skalar "Frequency" in die Liste des ausgewählten Skalars beizufügen.



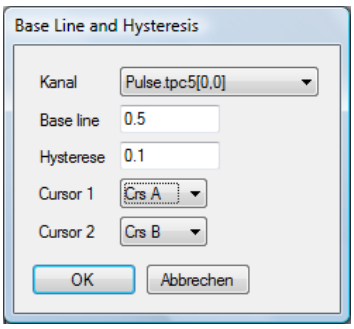
Es können auch weitere Funktionen ausgewählt und der Liste hinzu geführt werden. Jeder Eintrag wird in einer eigenen Spalte bei den Skalaren dargestellt.

Klicken Sie "OK", damit die eingetragenen Skalare, im Beispiel nur "Frequency", in der Skala Funktions-Tabelle ganz rechts hinzugefügt werden. Bei mehreren Einträgen in der Liste werden die Skalare der Reihe nach in der Tabelle von links nach rechts angehängt.

Tätigen Sie einen Rechtsklick auf die aktive Zeile in der Kolonne *Kurve* und klicken Sie "Base-Line, Hysterese und Cursor setzen..", damit sich der Dialog "**Base Line und Hysterese**" öffnet. Alternativ dazu doppelklicken auf den gewünschten Kanal.

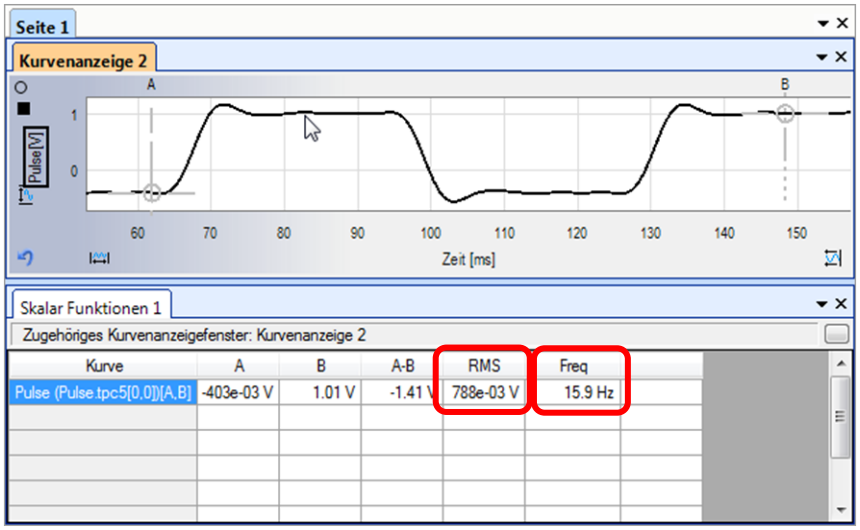


Im "**Base Line and Hysteresis**" Dialog kann nun die Messkurve, die Basislinie, die Hysterese und das Messfenster, definiert durch die Cursors, in dem zugehörigen Kurvenfenster bestimmt werden.



Die Frequenz wird nun bei der Basislinie von 0.5 V mit einer Hysterese von 0.1 V zwischen den Cursor A und B ermittelt.

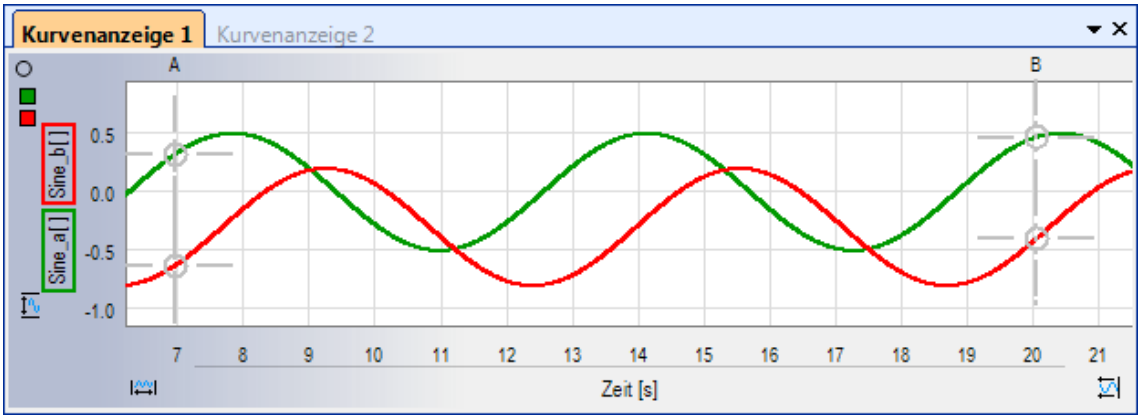
In der Tabelle Skalar Funktionen 1 können nun die Ergebnisse entnommen werden.

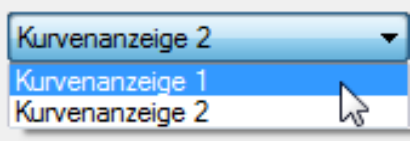


25.1.5 Beispiel 3: Phase

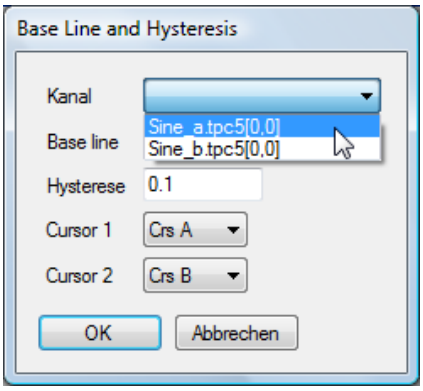
- *Phase* - Horizontale Skalar Funktion; mit Referenzkanal; mit Basislinie und Hysterese

In der folgenden Kurvenanzeige 1 sind zwei Kurven vorhanden, deren Phasenlage zu messen ist.

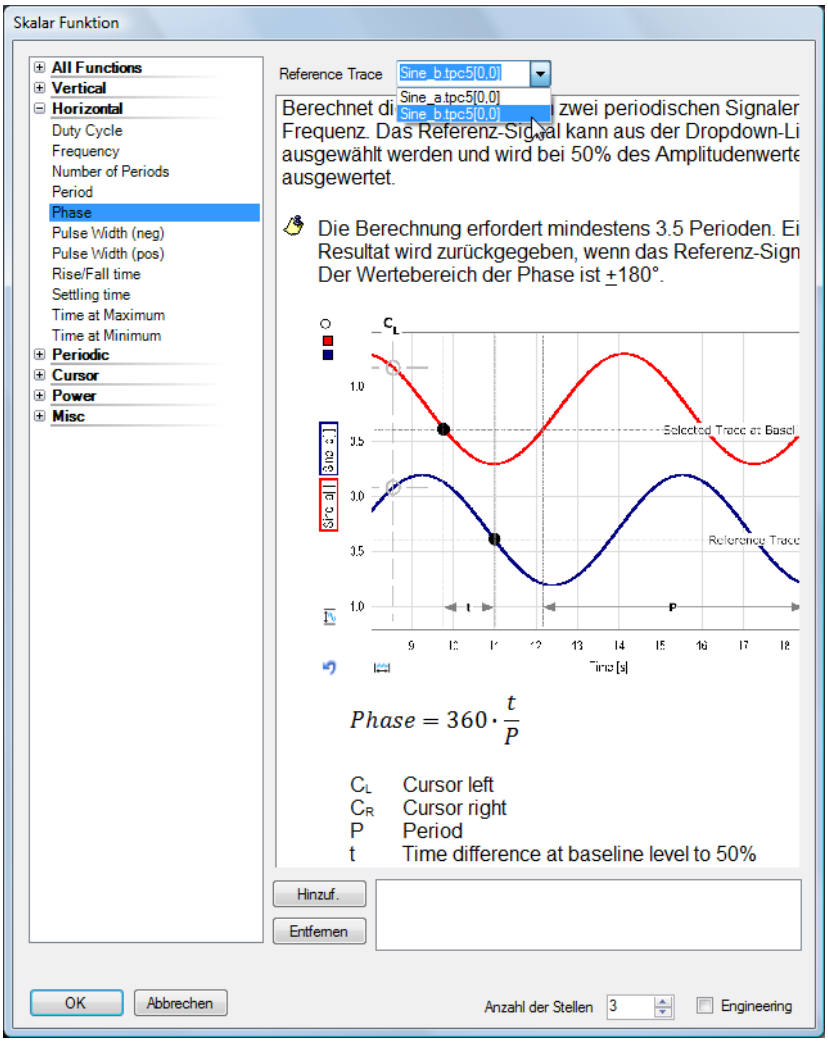




Zuerst muss die **Skalartabelle** auf die zugehörige **Kurvenanzeige eingestellt** sein. Wie weiter oben erklärt muss dazu bei der Auswahl "Zugehöriges Fenster" die entsprechende Kurvenanzeige ausgewählt werden. In diesem Fall Kurvenanzeige 1.



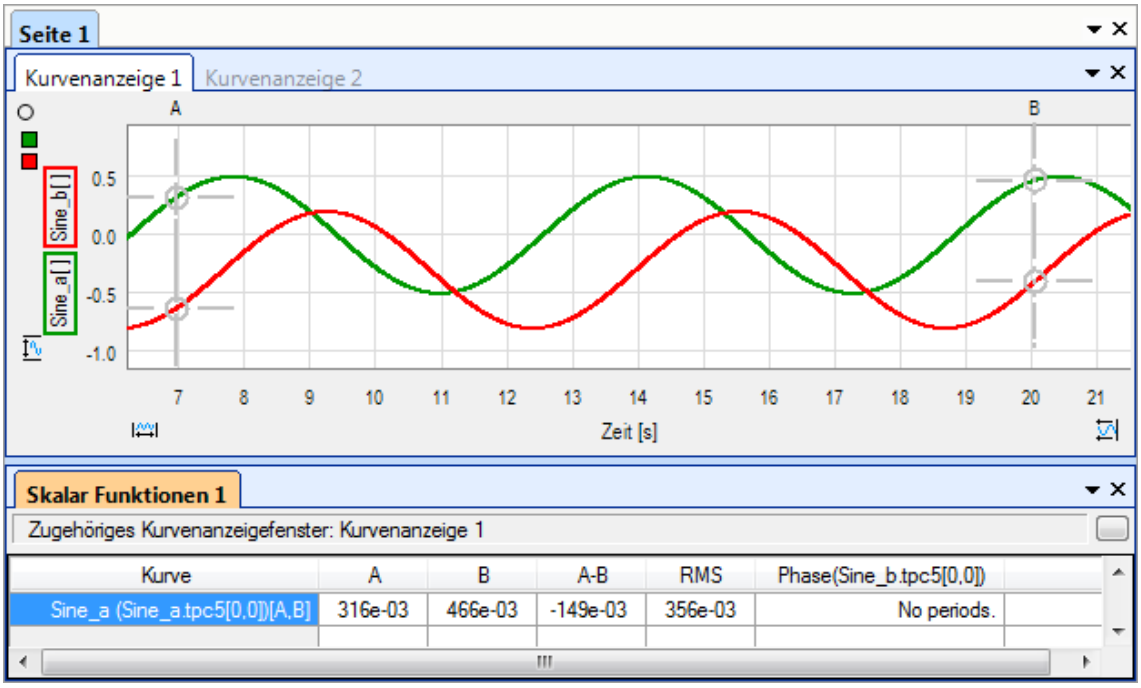
Sobald die Zugehörigkeit auf die "Kurvenanzeige 1" eingestellt ist, kann in der Skalartabelle auf deren Kurven zugegriffen werden.
Doppelklick auf die **erste Zeile** in der Kanal-Spalte. Das Fenster **Base Line and Hysteresis** wird geöffnet. Der Kanal *Sine_a* wird ausgewählt als Berechnungsquelle der ersten Zeile der Skalartabelle.

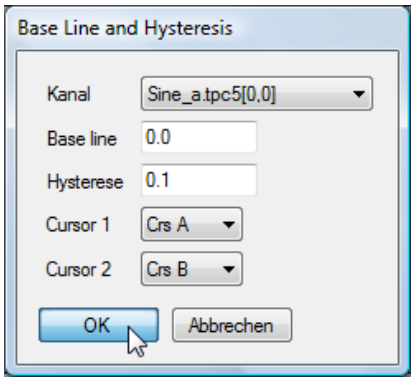


In diesem Beispiel soll der eingestellte Skalar "Frequency" durch "Phase" ersetzt werden. Dazu **Doppelklicken** Sie auf die Spalte mit der Skalar Funktion "Frequency". Wählen Sie in der **Liste "Horizontal" die Funktion "Phase"**. Selektieren Sie den **Referenzkanal "Sine_b"**, zu dem die Phase berechnet werden soll. Das ist der Bezugskanal zum bereits eingestellten Kanal "Sine_a".

Mit Klick auf **OK** wird die Einstellung übernommen und das Fenster **Skalar Funktion** geschlossen.

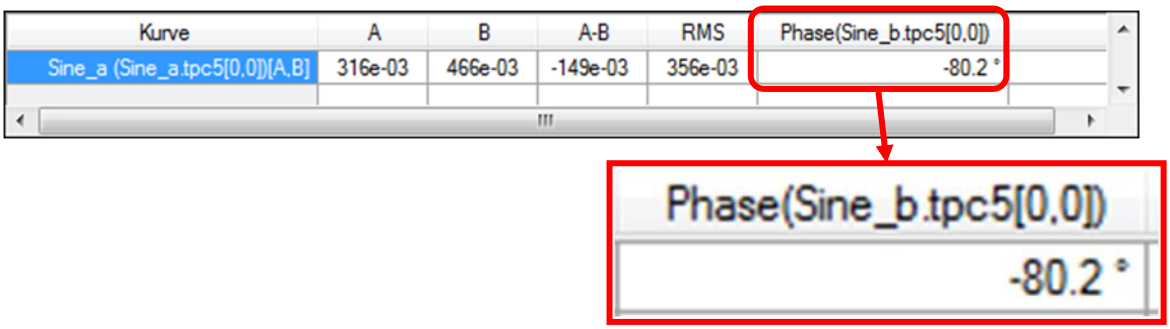
Die Phase gibt das Messergebnis "No periods" zurück. Das hat damit zu tun, dass entweder **nicht genügend Perioden** im Messfenster liegen oder dass der Basislinien-Pegel **keine Durchgangspunkte** mit der Kurve findet. Beim näheren Hinschauen stellt man fest, dass die Basislinie zu den Berechnungen an der Kurve *Sine_a* idealerweise bei 0 V liegt.



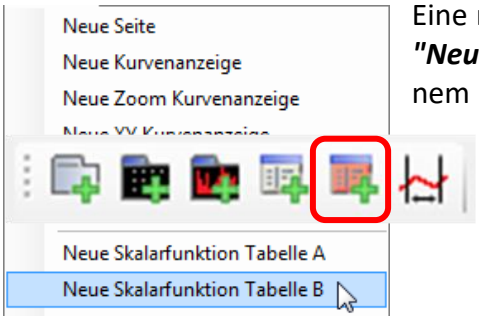


Im Fenster **"Base Line and Hysteresis"** werden die Werte, wie im nächsten Bild zu sehen ist, eingestellt.
Die Hysterese sollte nie auf null eingestellt werden, weil dadurch eventuell durch Rauschen oder Störungen zusätzliche Nulldurchgänge vorgetäuscht werden können. Klicken Sie **"OK"** zum Akzeptieren der Werte und Schliessen des Fensters.

Nun sind Basislinien-Durchgänge vorhanden und die Phase kann berechnet werden.

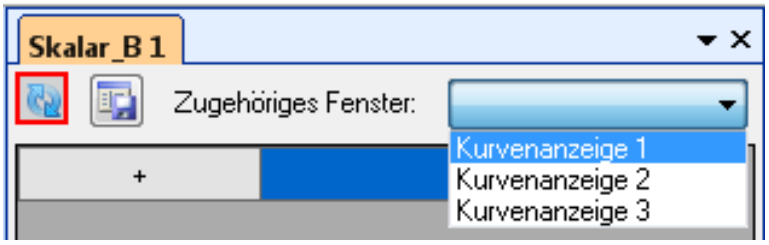


25.2 Skalar Funktionen Tabelle B



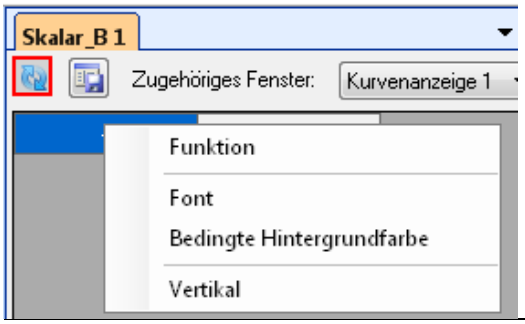
Eine neue Skalar Funktions Tabelle B kann im Menü ***"Ansicht" / "Neue Skalar Funktion Tabelle B"*** geöffnet werden oder mit einem Klick auf das Icon "**Neue Skalartabelle B**".

Das zugehörige Kurvenfenster, also die Quelle der zur Verfügung stehenden Kurven, kann im Dialog "**Zugehöriges Fenster"** ausgewählt werden. Hier werden alle verfügbaren Kurvenanzeigen aufgelistet.



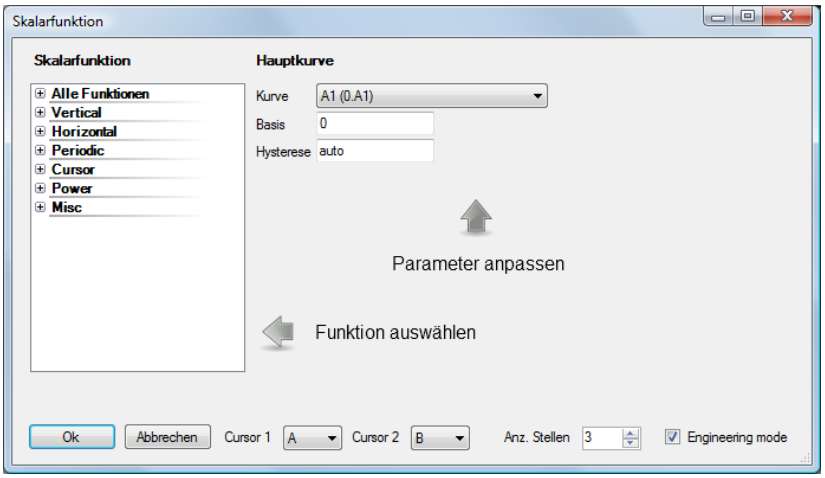
Falls gewünscht kann die neu erstellte Tabelle nun sehr **flexibel positioniert** werden. Durch Ziehen des Registerblattes ***"Skalar_B 1"*** lässt sie sich auf einer beliebigen Seite des aktiven Kurvenfensters platzieren.

25.2.1 Skalar Funktion hinzufügen



Um der Tabelle eine **Funktion hinzuzufügen** kann mit **Rechts-Klick** auf das **+** -Symbol das Kontextmenü geöffnet werden. Mit der Auswahl Funktion wird anschliessend das Fenster Skalar Funktion geöffnet. Alternativ dazu öffnet sich das Fenster Skalar Funktion direkt mit einem **Doppelklick** auf das **+** -Symbol in der untersten Linie bzw. in der Kolonne ganz rechts.

Die Kurven, die in der Skalar Funktions Tabelle per Zeile bzw. per Kolonne ausgewählt werden können, sind im zugehörigen Kurvenfenster vorhanden.

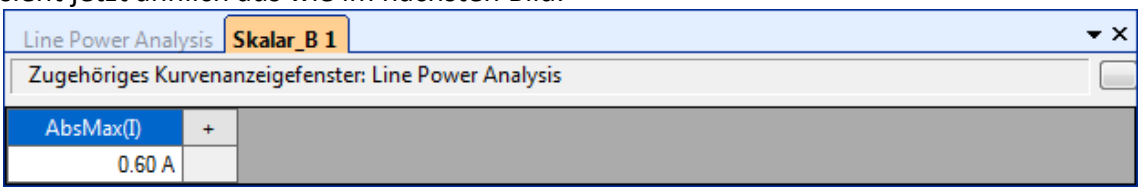


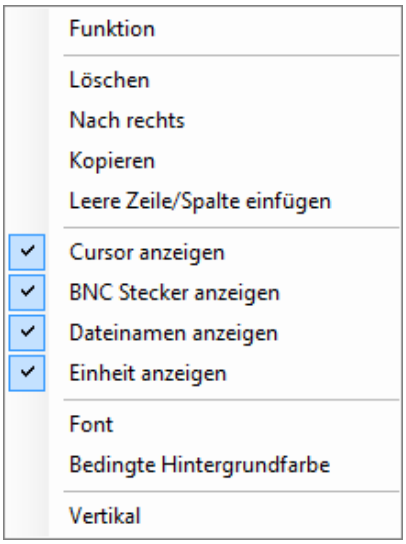
In diesem Fenster werden alle **Einstellungen** vorgenommen, die zu einer bestimmten **Skalar Funktion** gehören. Die Funktionen sind unterteilt in folgende Kategorien (wie bei Skalar-Tabelle A):

- **All Functions:** Alle Funktionen sind aufgelistet.
- **Vertical:** Aufgelistet sind diejenigen Funktionen, die vertikale Messungen an der Kurve durchführen.
- **Horizontal:** Aufgelistet sind diejenigen Funktionen, die horizontale Messungen an der Kurve durchführen. Diese Skalar Funktionen ermitteln typischerweise zeitabhängige Messgrössen wie Pulsbreite oder die Periodendauer. In der Regel benötigen diese Skalar Funktionen eine bestimmte Schwelle (Baseline) mit Hysterese.
- **Periodic:** Listet die periodischen Skalar Funktionen auf. Diese Skalare können nur auf periodische Signale angewendet werden. Sie benötigen als Zusatzinformation die Basislinie und Hysterese für die Ermittlung der ganzen Perioden.
- **Cursor:** Listet die Skalar Funktionen auf, die sich auf Cursor-Messungen beziehen.
- **Power:** Aufgelistet sind diejenigen Funktionen, die üblicherweise zu elektrischen Leistungsmessungen eingesetzt werden.
- **Misc:** Hier werden einige Spezial-Skalar Funktionen wie *Area* oder *Number of Triggers* aufgelistet, die keiner der oben erwähnten Kategorien zugeordnet werden können.

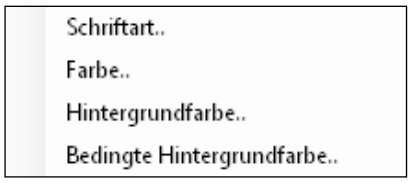
Nun wird die **Hauptkurve** für die **Berechnung** ausgewählt. Einige Funktionen wie die Scheinleistung (Apparent Power) beziehen sich noch auf eine zweite Kurve, die zugehörige Referenzkurve.

Die Einstellungen für die Basislinien mit Hysterese legen die Pegel zur Ermittlung von ganzen, abgeschlossenen Perioden fest.
Es muss weiter **definiert** werden, zwischen **welchen Cursor** die Skalar Funktion berechnet werden soll und in welchem Zahlenformat.
Mit Klick auf **OK** wird das Fenster geschlossen und die Einstellungen übernommen. Die Tabelle sieht jetzt ähnlich aus wie im nächsten Bild.





Ein **Rechts-Klick** auf die Zelle mit der **Skalar Funktion** in der Tabelle öffnet ein Dialogfenster. Das ermöglicht die Auswahl einer **anderen Skalar Funktion**, eine Kopie der aktuellen Funktion zu erstellen, die aktuelle Funktion zu löschen oder eine neue Spalte/Zeile einzufügen.
Weiter ist es möglich, die Darstellungsweise des Resultates anzupassen. Wahlweise können die Cursors, die Einheit, die Signaleingangsbuchse, und der Dateiname ein- oder ausgeblendet werden.
Die Auswahl ***Vertikal*** konvertiert die Tabelle von Zeilendarstellung nach **Spaltendarstellung**.
Die vertikale Darstellung eignet sich, wenn die Tabelle auf einer Seite des Kurvenfensters angedockt ist.



Ein Rechts-Klick auf eine Tabellenzelle, in der das Messergebnis dargestellt wird, öffnet ein Dialogfenster zur Konfiguration der Visualisierung des Resultates. Es können auch mehrere Zellen markiert werden und dann gemeinsam angepasst werden.

Ein Rechts-Klick auf die Zelle mit dem "+" öffnet einen Dialog zur Auswahl einer Skalar Funktion und bietet die Möglichkeit die Ausrichtung zu ändern (horizontal/vertikal).

Die Skalar Funktions Tabelle wird **normalerweise** ausschliesslich im **Single Trigger** Mode oder in einer Schleife einer Autosequenz mit entsprechender Wartezeit aktualisiert. Es besteht aber die Möglichkeit, Auto-Refresh der Tabelle einzuschalten.

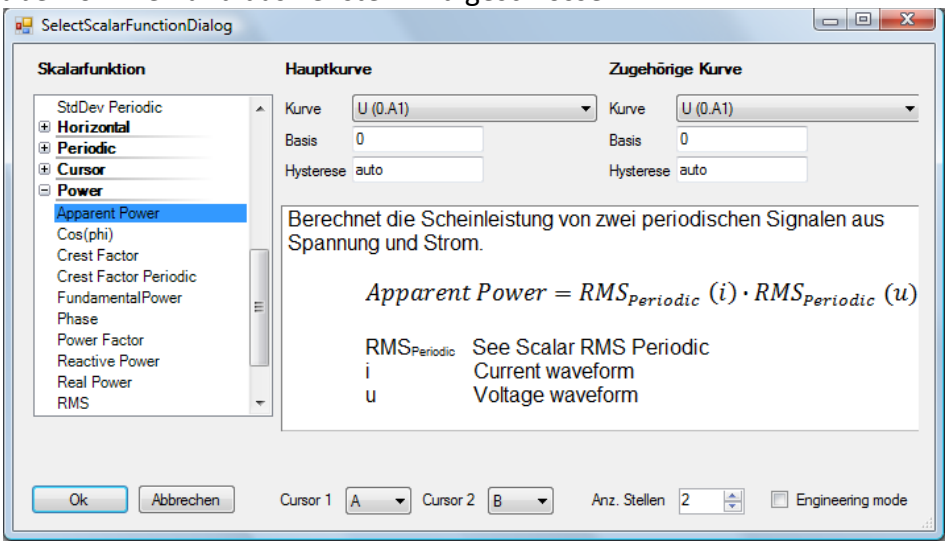
25.2.2 Beispiel 1: Apparent Power

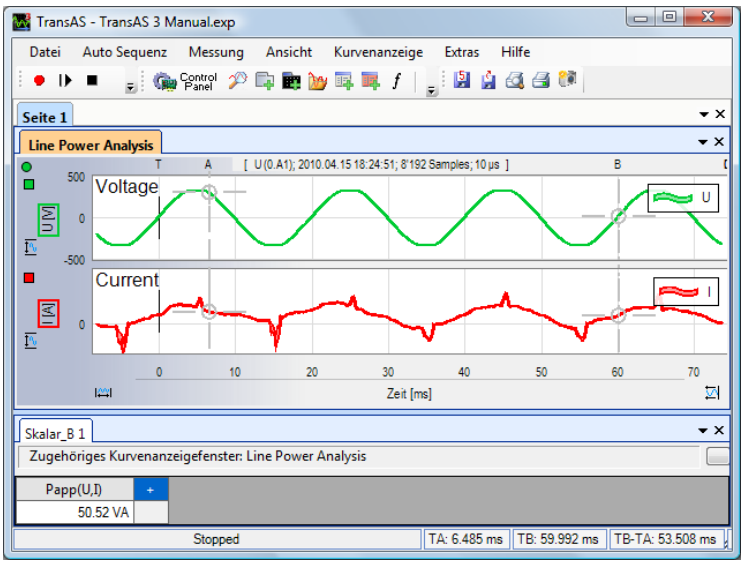
- *Apparent Power* (Scheinleistung) - Kategorie Power; mit zugehöriger Referenzkurve; mit Basislinie und Hysterese

Tätigen Sie einen Doppel-Klick auf die Skalartabelle, um entweder eine neue Funktion zu erstellen oder eine bestehende zu ersetzten. Öffnen Sie die Kategorie Power und wählen sie die Funktion "Apparent Power". Die Berechnungsformel wird im Fenster rechts der Liste angezeigt.

Wählen Sie die **Hauptkurve** aus (davon ausgehend, es sei eine Spannungskurve) mit Basislinie bei 0V und einer automatisch festgelegten Hysterese, welche 10% des Spitzen-Spitzenwert der Kurve entspricht.
Nun wählen sie die **Referenzkurve** aus (davon ausgehend, es sei eine Stromkurve) mit Basislinie bei 0V und einer automatisch festgelegten Hysterese, welche 10% (+/- 5%) des Spitzen-Spitzen-Wertes der Kurve entspricht.

Nach Grundeinstellung werden die Resultate im Nicht-Engineering-Mode dargestellt. In diesem Fall zwei Nachkommastellen beibehalten. Gegebenenfalls könnte hier die Anzahl Stellen angepasst und/oder auf Engineering-Format gewechselt werden. Mit Klick auf **OK** werden die Werte übernommen und das Fenster wird geschlossen.





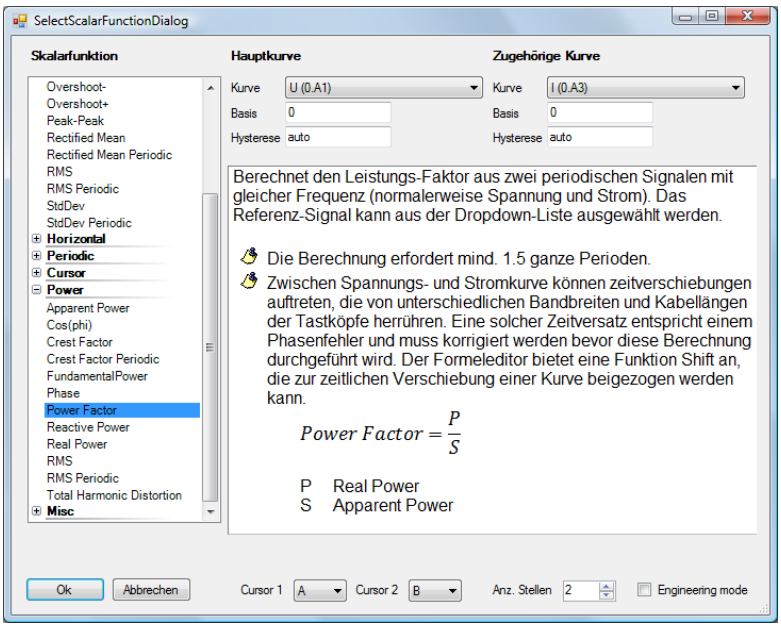
Die Scheinleistung wird nun zwischen den Cursor A und B berechnet und das Resultat in der Spalte "Papp(U/I)" dargestellt. Rechts von dieser Spalte wurde automatisch eine neue Spalte eingefügt.

Die Skalar Funktionen werden zwischen den gewählten Cursor berechnet (siehe auch Beschreibung der Skalar Funktionen). In der zugehörigen Kurvenanzeige können die gewählten Cursors so platziert werden, dass der gewünschte Kurvenbereich als Messfenster für die Berechnung selektiert ist.

25.2.3 Beispiel 2: Power Factor

- *Power Factor* - Kategorie Power; mit zugehörigem Referenzkanal und Basislinie mit Hysterese

Nun wird eine weitere Skalar Funktion hinzugefügt. Doppelklicken Sie auf das **+** -Symbol. Wählen Sie im Dialog "Skalar Funktion" die Kategorie "Power" öffnen und selektieren Sie die Funktion "Power Factor".
Das Zahlenformat ist immer noch Non-Engineering mit 2 Stellen, weil TranAX sich an die letzte Einstellung erinnert. Mit einem Klick auf OK werden die Einstellungen übernommen und das Fenster wird geschlossen.

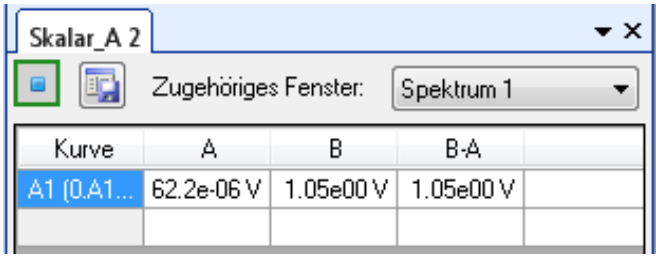


Der Leistungsfaktor aus den Kanälen *U(0.A1)* und *I(0.A3)* wird nun berechnet zwischen den Cursor A und B. Die Basislinien werden benötigt, um Anfang und Ende von ganzen Perioden zu definieren. Die Hysteresis ist auf *"Auto"* gestellt, was einem Wert von 10% (+/- 5%) der Spitzen-Spitzen-Werte der jeweiligen Kurve entspricht.

Einige Skalar Funktionen benötigen eine bestimmte Schwelle (Baseline) mit Hysterese.
Im Abschnitt Beschreibung der Skalar Funktionen werden alle Skalarfunktionen beschrieben und es ist ersichtlich, welche Einstellungsmöglichkeiten jede Funktion hat.

25.3 FFT Funktionen (Tabelle A/B)

Wenn beim Öffnen einer neuen Skalar-Tabelle (A oder B) ein **Spektrum-Fenster** den **Fokus** hat, wird automatisch die Skalar-Tabelle mit dem Spektrum als **"zugehörigem Fenster"** gebildet. Dabei werden auch automatische die vorhandenen Spektrum-Kurven in der Tabelle eingetragen.



Prinzipiell ist die Bedienung zu den Skalar-Tabellen von Spektren gleich, wie bei Skalar-Funktionen zu den zeitbezogenen Signalen. Einzig die Liste der möglichen **Funktionen** ist auf die Berechnung von **Spektrum-Parametern zugeschnitten**.

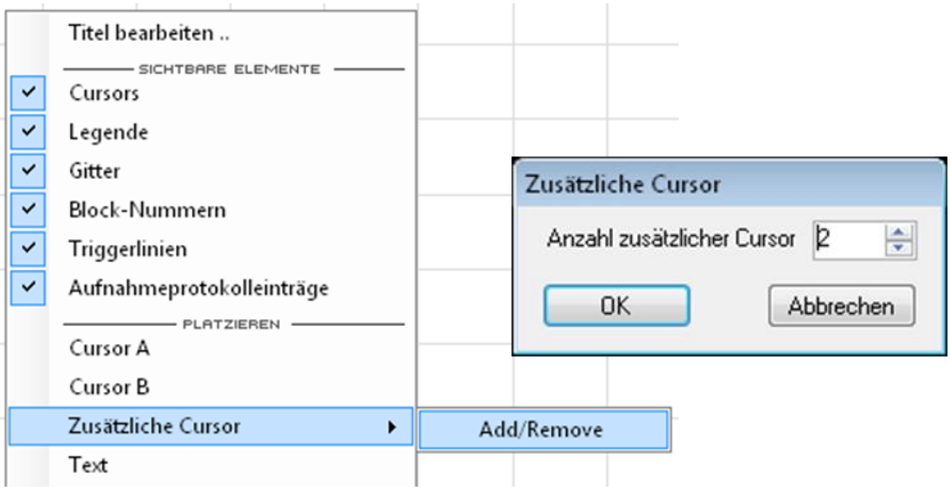
Folgende Spektrum-Parameter können berechnet werden:

- Cursor Amplitude
- Cursor Delta Amplitude
- Cursor Delta Position
- Cursor Position
- Cursor Ratio Amplitude (dB)
- Frequency at Maximum
- Maximum (zwischen zwei Cursors)

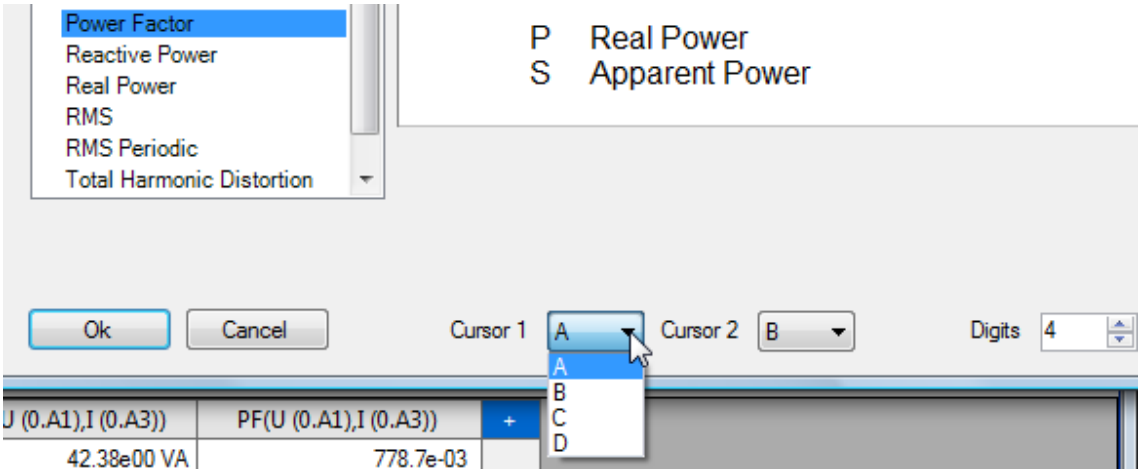
In den Skalar-Tabellen kann unter "Zugehöriges Fenster" von einem Zeitfenster (Kurvenanzeige) zu einem Spektrum (oder umgekehrt) gewechselt werden. Danach müssen jedoch die entsprechenden **Kurven neu via Drag & Drop** in die Tabelle gezogen werden. Auch müssen in der Regel die **Funktionen angepasst werden** ("Mean" z.B. kann in einem Spektrum nicht gerechnet werden).

25.4 Zusätzliche Cursors

Falls mehr als ein Messbereich, also **zusätzliche Cursor** benötigt werden, fügt man diese mit Rechtsklick im Kurvenfenster und der Auswahl *Zusätzliche Cursor* hinzu. Zusätzlich heisst zusätzlich zu den Cursor A und B.



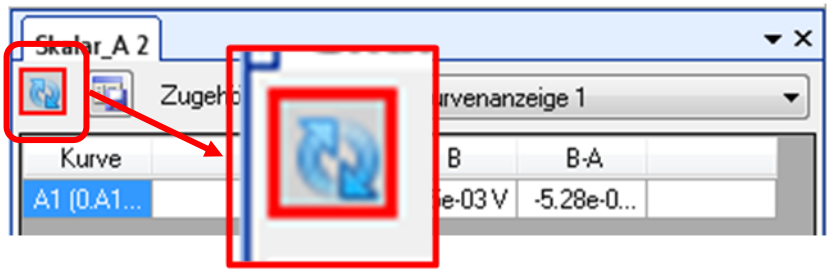
Falls die Anzahl zusätzlicher Cursors 2 ist, so werden anschliessend 4 Cursors im Kurvenfenster zur Verfügung stehen, C und D kommen zu A und B hinzu. Diese können dann im Fenster **Skalar Funktion** auch ausgewählt werden.



25.5 Auto-Refresh der Skalar Funktion Tabelle

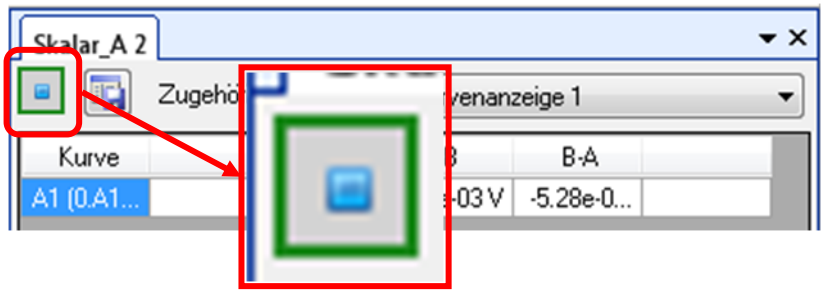
Im **Single Shot Mode** werden die Skalar-Werte der Skalar Funktions-Tabelle nach jeder Datenerfassung **aufdatiert**. Falls Single Shot Mode ausgeschaltet ist, werden die Skalar-Werte nur dann aufdatiert, wenn der **Schalter Auto-Refresh** eingeschaltet ist (grüner Rahmen).

Im folgenden Bild ist **Auto-Refresh ausgeschaltet (roter Rahmen)**.



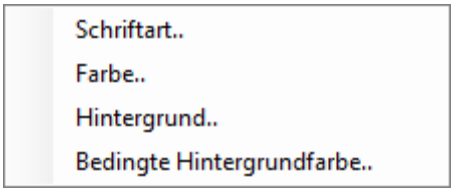
Da das Auto-Refreshing der Skalartabelle eine grosse **Auswirkung** auf die **Systemleistung** hat empfiehlt es sich, Auto-Refresh nur dann einzuschalten, wenn es wirklich benötigt wird. Falls die Update-Rate der Kurvenanzeige wichtiger ist als die Berechnungen der Skalar Funktionen sollte Auto-Refresh ausgeschaltet werden.

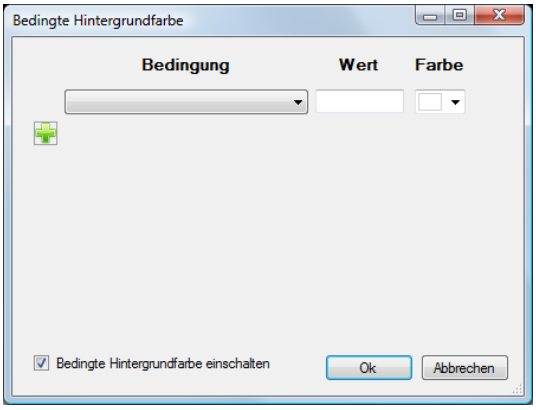
Im folgenden Bild ist **Auto-Refresh eingeschaltet (grüner Rahmen)**.



25.6 Bedingte Hintergrundfarbe

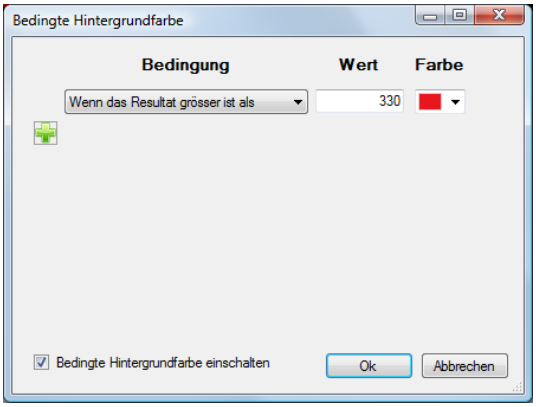
Zu jeder **Skalar Funktion** können **Bedingungen definiert** werden, die den Anwender bezüglich bestimmter Skalar-Werte **visuell** informieren. Mit einem **Rechts-Klick** auf den Wert der Skalar Funktion und der Auswahl "Bedingte Hintergrundfarbe" öffnet sich das entsprechende Fenster.



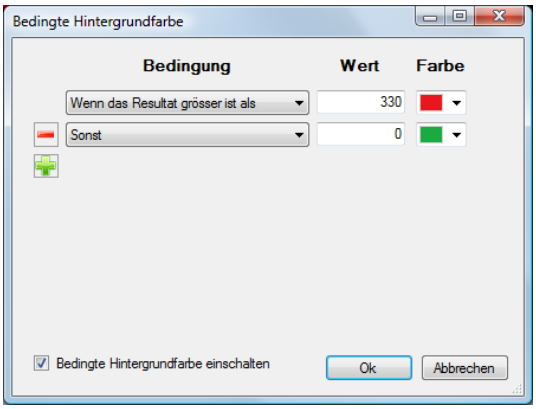


Für jedes **Feld** können **mehrere Bedingungen** definiert werden. Es stehen die folgenden Bedingungen zur Verfügung:

Wenn das Resultat grösser ist als
Wenn das Resultat kleiner ist als
Sonst
Funktion hat einen Fehler
Deaktiviert



Es wird nun die Bedingung definiert, dass die Hintergrundfarbe rot wird beim Überschreiten des Wertes von 330V.



Eine weitere Bedingung wird mit der Schaltfläche hinzugefügt.

Mit der zweiten Bedingung wird nur gesagt, dass die Hintergrundfarbe grün sein soll, falls die erste Bedingung nicht erfüllt ist.

Es wird empfohlen, die Werte in auf- oder absteigender Reihenfolge zu listen.

Die bedingte **Hintergrundfarbe** muss noch **aktiviert** und das Fenster mit **Ok** geschlossen werden. Die Skalar Funktion mit der bedingten Hintergrundfarbe wird nun entsprechend dem Skalar-Wert ausgewertet:

A(A1 (0.A1))	A(I (0.A3))	PP(A1 (0.A1)) [A,B]	+
23.1766 V	-0.0575 V	340e00 V	

Spannung ist **über 300V**, die Zelle wird **rot** gefärbt.

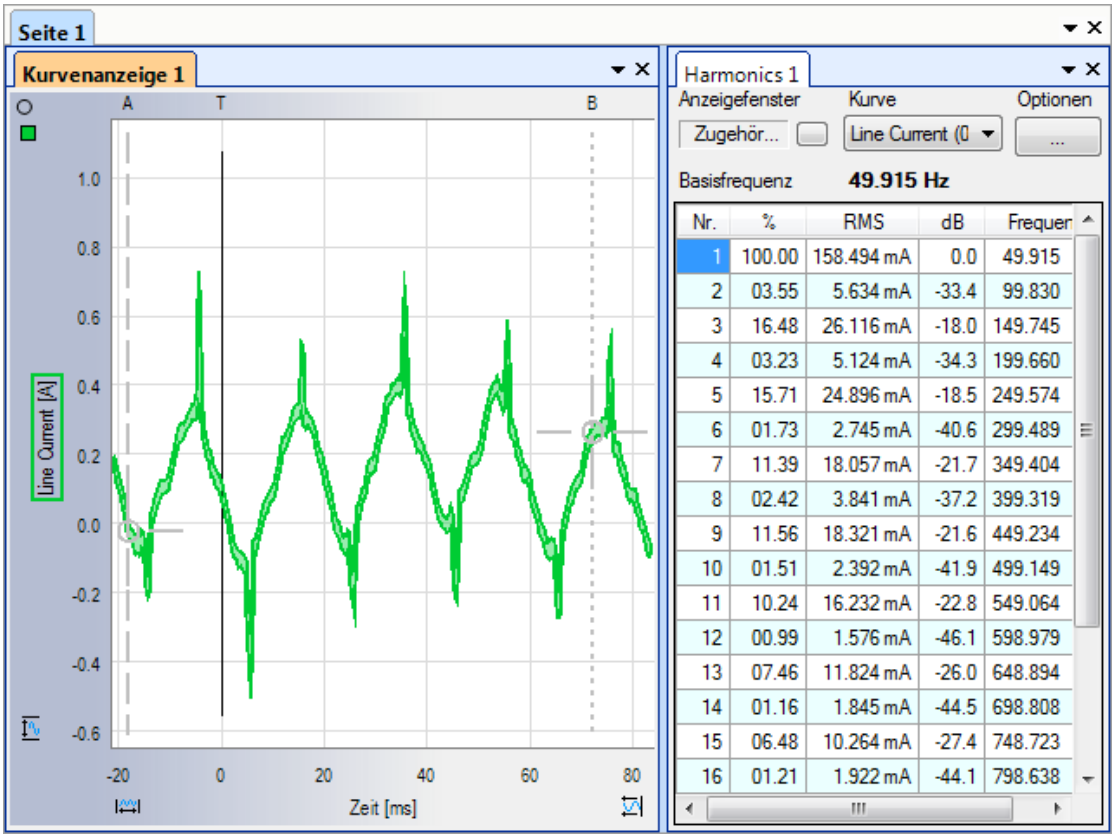
A(A1 (0.A1))	A(I (0.A3))	PP(A1 (0.A1)) [A,B]	+
23.1766 V	-0.0575 V	215e00 V	

Bei einer Spannung **unter 300V** bleibt diese **grün**.

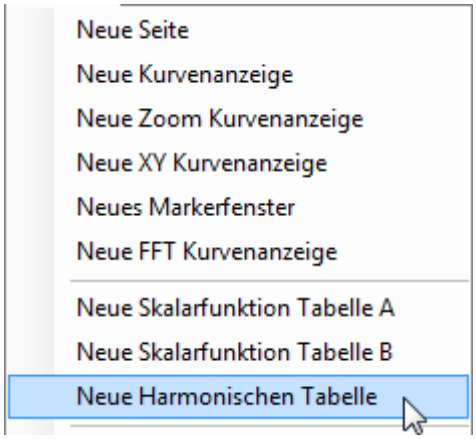
26 Tabelle der Harmonischen

Ein **periodisches Signal der Zeitebene** kann mit Hilfe der **Fourier Analyse** in die Frequenzebene konvertiert werden. Dabei werden die Amplituden bezogen auf die Frequenz aufgeschlüsselt in die Grundwelle und die Harmonischen.

Die Tabelle der Harmonischen kann beliebig neben der Kurvenanzeige platziert oder als Registerkarte beibehalten werden. Für weitere Information betreffend der "Window Dock-in" Technik siehe auch Abschnitt TranAX Übersicht.

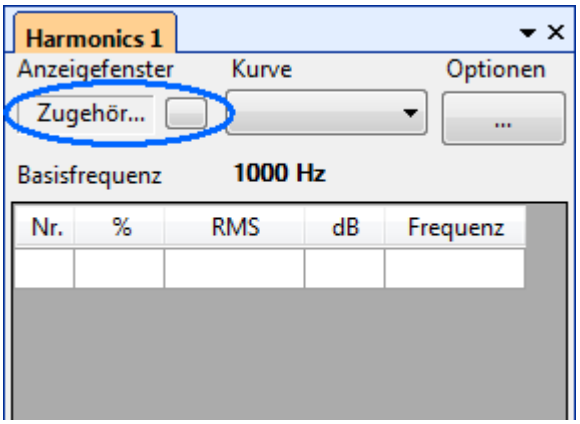


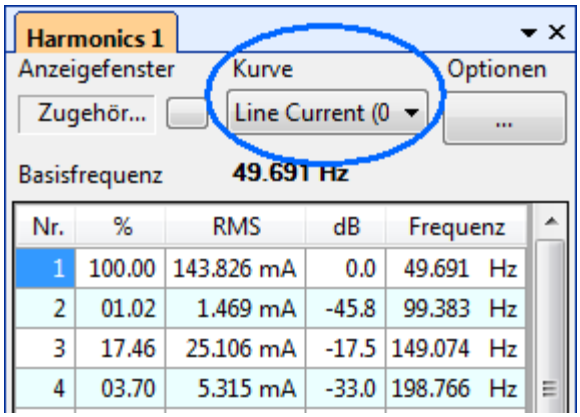
Nr.	%	RMS	dB	Frequer
1	100.00	158.494 mA	0.0	49.915
2	03.55	5.634 mA	-33.4	99.830
3	16.48	26.116 mA	-18.0	149.745
4	03.23	5.124 mA	-34.3	199.660
5	15.71	24.896 mA	-18.5	249.574
6	01.73	2.745 mA	-40.6	299.489
7	11.39	18.057 mA	-21.7	349.404
8	02.42	3.841 mA	-37.2	399.319
9	11.56	18.321 mA	-21.6	449.234
10	01.51	2.392 mA	-41.9	499.149
11	10.24	16.232 mA	-22.8	549.064
12	00.99	1.576 mA	-46.1	598.979
13	07.46	11.824 mA	-26.0	648.894
14	01.16	1.845 mA	-44.5	698.808
15	06.48	10.264 mA	-27.4	748.723
16	01.21	1.922 mA	-44.1	798.638



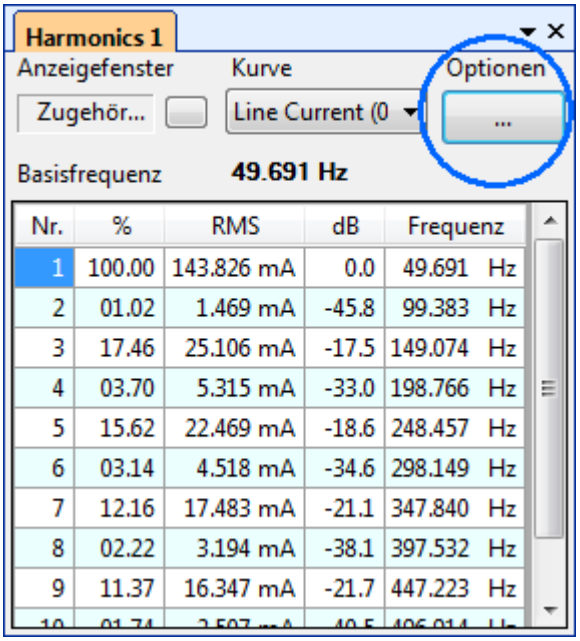
Eine neue Tabelle der Harmonischen wird im Menü ***"Ansicht" / "Neue Harmonischen Tabelle"*** geöffnet.

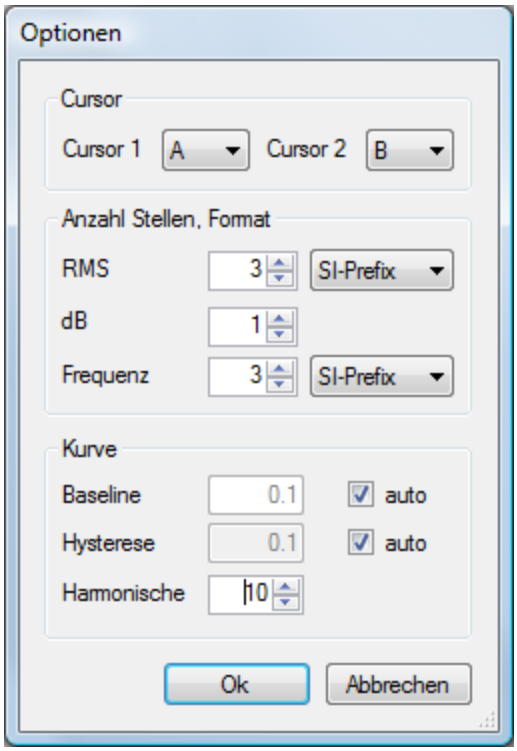
Eventuell muss dieser Tabelle zuerst eine **Kurvenanzeige zugeordnet** und anschliessend daraus die zu analysierende Kurve ausgewählt werden.





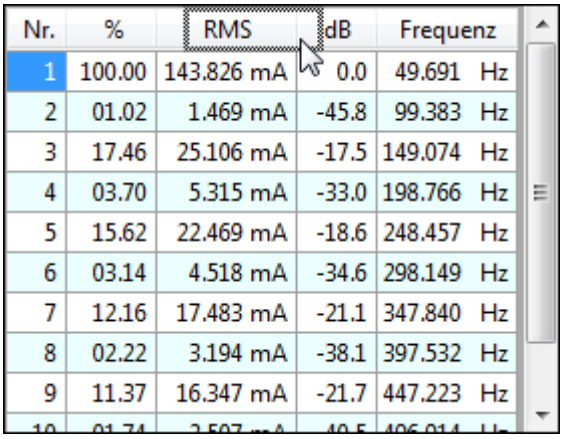
In den Optionen kann eingestellt werden, zwischen **welchen Cursors**, bei welcher Schwelle (Baseline) und Hysterese die Kurve **ausgewertet** werden soll.





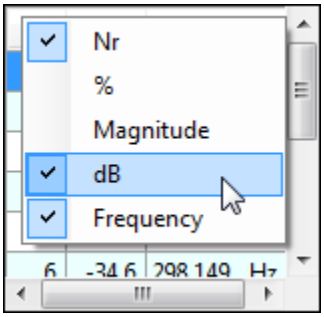
- **Cursor:** Die Harmonischen werden zwischen den Cursor A und B berechnet. Zur Berechnung müssen mindestens zwei Signalperioden zwischen den Cursor liegen.
- **Anzahl Stellen, Format:** Die Formate werden gewählt, wie die Resultate dargestellt werden sollen.
- **Kurve:** Die Schwelle (Baseline) und Hysterese sind für die Bestimmung der Periodendauer/Frequenz erforderlich. Falls **Auto** angewählt ist, wird die Basislinie in die Mitte des Signales und die Hysterese auf 10% (+/- 5%) des Spitzen-Spitzen-Wertes gelegt. Im Eingabefeld **Harmonische** wird eingetragen bis zu welcher Nummer die Harmonischen berechnet werden sollen (Bereich 1 ... 100).

Die Tabelle kann, hinsichtlich der Reihenfolge der Kolonnen und welche Kolonnen sichtbar sein sollen, angepasst werden.

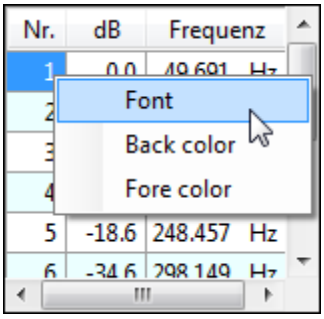


Nr.	%	RMS	dB	Frequenz
1	100.00	143.826 mA	0.0	49.691 Hz
2	01.02	1.469 mA	-45.8	99.383 Hz
3	17.46	25.106 mA	-17.5	149.074 Hz
4	03.70	5.315 mA	-33.0	198.766 Hz
5	15.62	22.469 mA	-18.6	248.457 Hz
6	03.14	4.518 mA	-34.6	298.149 Hz
7	12.16	17.483 mA	-21.1	347.840 Hz
8	02.22	3.194 mA	-38.1	397.532 Hz
9	11.37	16.347 mA	-21.7	447.223 Hz

Mit Hilfe von Drag & Drop kann die **Reihenfolge** der einzelnen Spalten **geändert** werden.



Via **Rechts-Klick** auf der Titellinie lassen sich Kolonnen ein- und ausschalten so dass nur die gewünschten **Informationen** angezeigt werden.

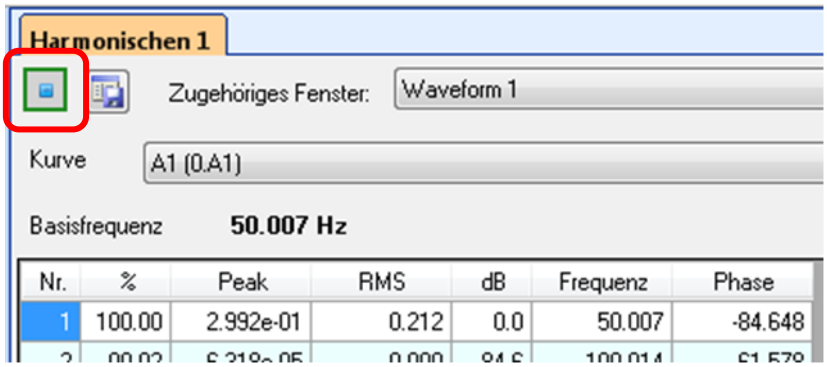


Rechts-Klick innerhalb einer **Zelle** oder einer Auswahl von Zellen, um die Schrift und Farben zu editieren.

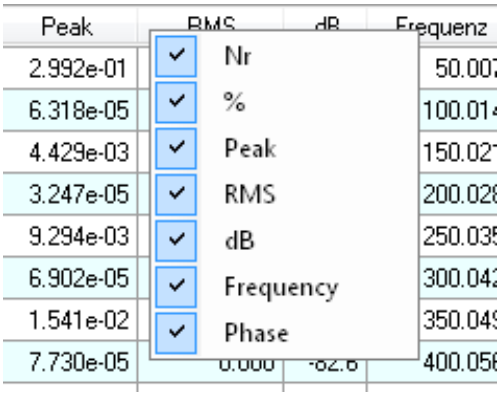
26.1 Erweiterte Harmonische Tabelle

Export Funktion: Durch Klicken des Knopfs kann die gesamte Tabelle im Text-Format exportiert werden. Diese Datei kann anschliessend z.B. in Excel wieder importiert und gegebenenfalls weiterbearbeitet und verwendet werden.

Aktualisierung während der Messung: Mit dem Knopf oben Links kann die Aktualisierung zur Laufzeit ein- oder ausgeschalten werden, gleich wie bei den Skalartabellen.



Beim hier dargestellten Zustand ist die Aktualisierung eingeschalten.

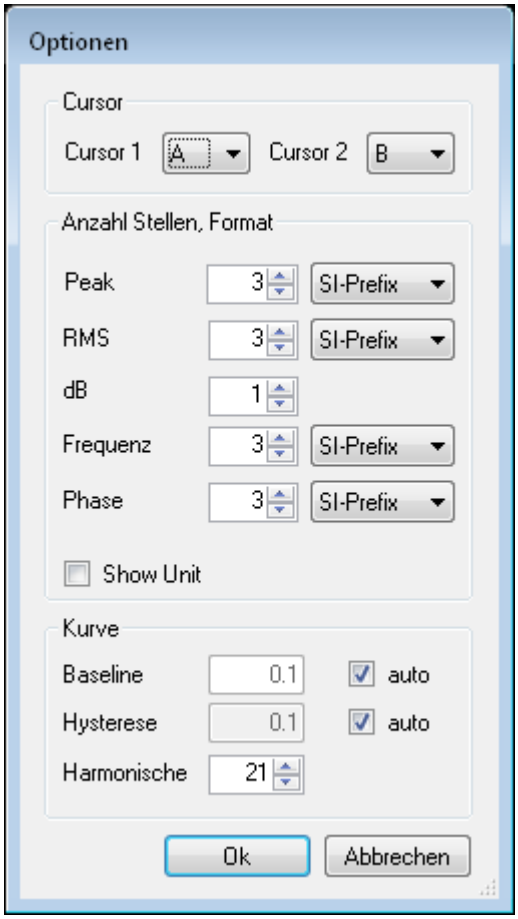


Mit einem **Rechtsklick** auf die obere Titelleiste können die einzelnen **Spalten ein- oder ausgeschalten** werden. Es werden nur die Spalten mit dem Häklein angezeigt.

Die Messwerte für **"Peak"** und die **"Phase"** der Harmonischen können nun ebenfalls in der Tabelle dargestellt werden.

Es ist darauf zu achten, dass die **Cursors richtig platziert** wurden. Um die Harmonischen berechnen zu können, werden **mindestens zwei ganze Perioden** der Grundwelle benötigt!

Bei den **"Optionen"** können die folgenden Einstellungen getätigt werden:

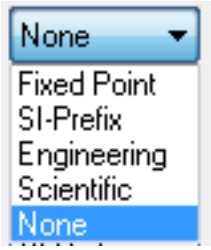


Auswahl der zu verwendenden Cursors: Als Standard sind A und B eingestellt.

Anzahl Stellen und Format für jede einzelne Spalte kann die Anzahl der Stellen und die Formatierung der Zahl gewählt werden.

Show Unit: Mit dieser Option kann bestimmt werden, ob die Einheiten bei den Resultat-Werten angezeigt werden oder nicht. Für Exporte zur weiteren Bearbeitung in externen Programmen wird diese Option mit Vorteil deaktiviert, so dass nur reine Zahlenwerte gespeichert werden.

Kurve: Baseline und Hysterese kann entweder manuell vorgegeben oder (wie empfohlen) automatisch ermittelt werden. Stark gestörte Signale erfordern oft eine manuelle Einstellung, vor allem der Hysterese.
Die Anzahl der zu berechnenden Harmonischen kann ebenfalls eingegeben werden.



Zu jeder Resultat-Spalte kann die Darstellung des Zahlenwerts definiert werden. Unterschieden wird zwischen **Fixed Point**, **SI-Prefix**, **Engineering**, **Scientific** und **None**.

Beispiel: Messwert für "Peak" beträgt 23'070.20897 V

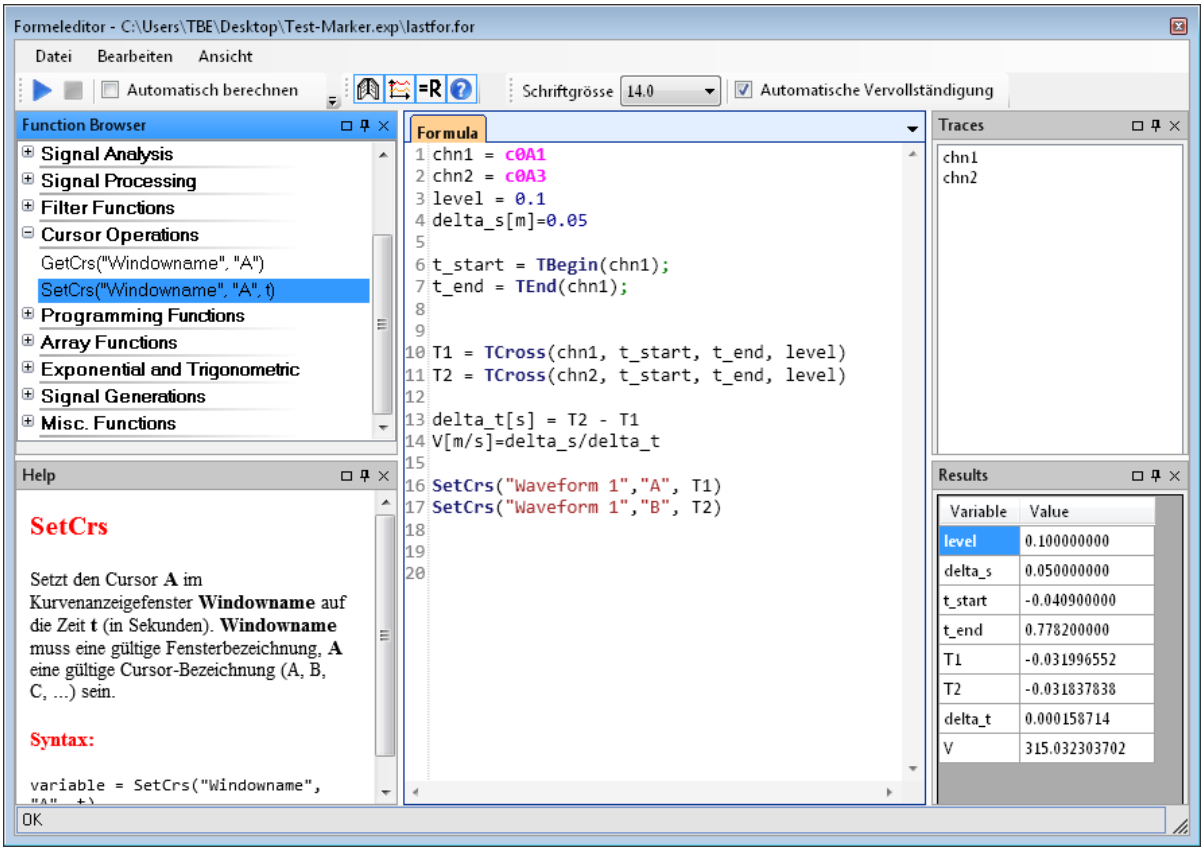
Format	Stellen	Anzeige
Fixed Point	3	23'070.209 V
SI-Prefix	3	23.070 kV
Engineering	3	23.1e03 V
Scientific	3	2.307e04 V
None	3	2.31E+04 V

27 Formeleditor

TranAX bietet mit dem integrierten Formeleditor eine grosse Fülle von hilfreichen Funktionen zur Analyse der erfassten Kurven und deren weitere Aufbereitung. Berechnung von neuen Kurven und Messwerten können ohne grossen Aufwand durchgeführt werden.

Mit den über **100 mathematischen Funktionen** und Operatoren kann nahezu jede praxisorientierte Berechnung durchgeführt werden.
Eine komplette Liste mit Funktionen ist im Anhang gelistet.

Der Formel-Editor bieten den Komfort aktueller Programmierumgebungen: Syntaxhervorhebung (Syntax-Highlighting), automatische Vervollständigung, Anwenderspezifische Schriftgrösse, etc. Neben dem Haupt-Formel-Textfeld können via "Datei/Funktion in separate Seite lade.." weitere Tabs mit andern Formel- oder Funktions-Dateien eingefügt werden. Dies dient vor allem dazu, bestehende Formeln oder Funktionen (nebst den Haupt-Formeln) zu inspizieren, vergleichen, ändern usw.



Der Formeleditor besteht aus den folgenden Komponenten:

- Die **Berechnungen** bzw. **Formeln** werden im Textfeld in der Mitte **"Main Formula"** eingegeben. Jede Linie repräsentiert eine Funktion und jede Funktion wird in der Regel durch einen Zeilenumbruch getrennt.
- **Rechts** sind zwei Spalten für die **Resultate**, unten die Skalar-Werte, oben die berechneten Kurven. Die Kurven können dann mittels Drag & Drop in ein Kurvenanzeige-Fenster gezogen werden.
- Skalar-Werte unter "Results" können auch in Text-Feldern in Kurvenanzeigen zur Ansicht gebracht werden.
- Links oben befindet sich eine Liste mit allen **verfügbaren Operatoren und Kanälen**. Sie können mittels Doppelklicks zum Textfeld hinzugefügt werden. **Unten** ist die Hilfe zu den einzelnen Funktionen eingeblendet.

27.1 Mit dem Formeleditor arbeiten

Zur Dokumentation von Formeln können Kommentare eingefügt werden. Diese sind durch einen Strich-Punkt ";" gekennzeichnet. Alles was nach einem Strich-Punkt auf einer Linie geschrieben ist, bleibt unberücksichtigt und wird als Kommentar interpretiert.
Zusätzlich kann in einem Kommentar ein Wort(teil) markiert und darauf ein Link zu einer Datei angefügt werden. Diese Datei wird dann (unsichtbar) in die Formel-Datei gespeichert. Sie kann eine ausführliche Beschreibung oder Anleitung zu den Formeln enthalten und kann wieder via Rechts-Klick über der entsprechenden Stelle geöffnet werden. Es können PDF-, Text- und Bilddateien angefügt werden.

Die einzelnen Berechnungen werden im Textfeld nachfolgendem Muster eingegeben:

```
name[Einheit] = Ausdruck
```

name ist ein **Variablen-Name**, welcher das Resultat benennt. Das Resultat kann in nachfolgenden Formeln eingesetzt werden, um weitere Berechnungen damit durchzuführen. Die *Einheit* dient dazu, berechneten Kurven eine ***Einheit* zuzuweisen**. Diese muss aufgrund der Einheiten der Kurven, welche in die Berechnung einfliessen, sowie allfälliger Skalierungen bestimmt werden. Beispiel: C0A1 misst eine Spannung in Volt, C0A2 messe einen Strom in Ampere. Dann wäre folgende Berechnung korrekt:

```
Leistung[mW] = c0A1 * c0A2 * 1000
```

Der Variablen-Name *"name"* kann im Prinzip beliebig gewählt werden. Einzig sogenannte Schlüsselwörter (z.B. Cos, Sinus, If, For, to, as, Pi usw.) sind nicht erlaubt und erzeugen eine Syntax-Fehlermeldung.
Auf der **rechten Seite** des **Gleichheitszeichens** können **beliebige Ausdrücke** stehen, welche die **Grundoperationen** Addition und Subtraktion, Multiplikation und Division sowie Potenzfunktionen beinhalten. Das Vorzeichen-Minus wird ebenfalls akzeptiert (z.B. –4 oder –c0A1). Multiplikation und Division haben eine höhere Priorität als Addition und Subtraktion. Klammern können beliebig gesetzt werden. Die **Operanden** können **Zahlen**, **Kanäle**, **Kurven** aus **TPC5- oder TDP-Dateien, Strings** (Zeichenketten)**, Arrays** oder vorher berechnete **Resultate** sein.
In der Regel werden die internen Berechnungen in doppelter Präzision (64Bit, 15 Dezimal-Stellen, Exponent > 300) durchgeführt. Gespeicherte Resultat-Kurven werden jedoch nur als 32Bit-Zahl abgelegt.

Kanäle werden mit einem "**c**", der **Geräte-Nummer** und der **Eingangsbezeichnung** angegeben (z.B.: C0A1, C0B3). Für Aufnahmen mit mehr als einem Block (Multiblock oder ECR) muss die **Blocknummer** angegeben werden (z.B. C0A1.2), andernfalls wird das Signal vom Block 0 berechnet. Mit einem **Apostroph (')** und einer Ziffer von 1 bis 2 kann ein Marker des Kanals ausgewählt werden:

```
Marker_1 = c0A1'1
Marker_2 = c0B3'2
```

Marker-Resultate werden durch die Analog-Werte 0 und 1 repräsentiert.

Kurven aus Dateien: Eine Datei wird mit deren Namen eingegeben:

```
File("dateiname.tpc5", 1)
```


Um die gewählte Datei nachträglich zu ersetzen, kann mit der rechten Maustaste auf den Dateinamen geklickt werden. Im Kontextmenü anschliessend "Set Filename.." klicken und mit dem Datei-Dialog die entsprechende neue Kurven-Datei auswählen.

Der **Index** repräsentiert die **Kurven-Nummer** in der Datei (0 entspricht der ersten Kurve, dieser Index ist nicht zu verwechseln mit den Kanal-Nummern!). Im oberen Beispiel wird also die zweite Signalkurve der Datei für die Berechnung verwendet.

Wenn über dem Index einer File-Funktion die rechte Maus-Taste gedrückt wird, werden alle Kurven in der Datei mit ihrem Namen der Reihe nach aufgelistet.

Sollen anstelle der Analog-Daten die Marker-Daten aus der Datei geholt werden, kann dies durch einen dritten Parameter angegeben werden:

```
File("Dateiname", Index).Block'Marker
```

Der Wert für den Parameter "Marker" muss entweder 1 oder 2 sein. Beispiel:

```
Signal = File("crash.tpc5",0)'1
```

Wenn kein Pfad mit dem Dateinamen angegeben wird, sucht das Programm die Kurve im Verzeichnis **DATA** des **aktuellen Experiments**. Für eine Datei in einem anderen Verzeichnis innerhalb des aktuellen Experiments muss nicht die gesamte Pfadbezeichnung (z.B. "C:\User\USERNAME\Documents\TranAX\EXPERIMENT.exp\ref\name.TPC5") angegeben werden. Es genügt die Bezeichnung "..\ref\name.TPC5".

Was nach einem **Semikolon (;)** auf einer Zeile steht, wird vom Formel-Interpreter bis zum Zeilenende ignoriert. Auf diese Weise können erklärende **Kommentare** eingegeben werden.

27.2 Cursors platzieren

Die Funktion ***SetCrs****("Fensternamen", "A", t)* **platziert Cursors** in der Kurvenanzeige. Die resultierende Position der Cursor im Fenster bezieht sich auf die X-Achse und hängt von der eingestellten X-Achsen-Skalierung ab (relativ zum Start, relativ zum Triggerzeitpunkt und Samples). Diese Funktion ist nützlich, wenn mit Skalar Funktionen z.B. die Fläche unter einem Impuls bestimmt werden soll. Dazu kann der Puls-Anstieg und -Abfall mittels *TCross* gesucht werden, um danach die Cursors mittels *SetCrs("...","A")* und *SetCrs("...","B")* auf die **gefundenen Puls-Grenzwerte** zu setzen.

Weiteres Beispiel:
Die Start- und End-Zeit der Kurve bestimmen

```
t_start = TBegin(c0A1)
t_end = TEnd(c0A1)
```

Bei beiden Kurven den ersten Anstieg auf über 1V suchen

```
T1 = TCross(c0A1, t_start, t_end, 1)
T2 = TCross(c0A2, t_start, t_end, 1)
```

Die beiden Cursors bei den gefundenen Durchgängen platzieren

```
SetCrs("Kurvenanzeige 1", "A", T1)
SetCrs("Kurvenanzeige 1", "B", T2)
```

27.3 String Variablen

String Variablen z.B. "Kurvenanzeige 1" oder "A" können mit den entsprechenden Funktionen deklariert werden. Beispiel:

```
value$ = "test.tpc5"
```

Das $-Zeichen in einer String-Variablen dient nur zur besseren Unterscheidung. Eine String-Variable kann auch ohne $-Zeichen vorgegeben werden.

Mehrere Strings können miteinander verknüpft (aneinandergereiht) werden:

```
name$ = "test"
extension$ = ".tpc"
version = 5
value$ = name$ + extension$ + version
```

Bei String-Operationen werden vorgegebene Zahlen automatisch als String interpretiert.
Zum Beispiel muss hier *version=5* nicht als *version="5"* geschrieben werden (die Funktion *version=2*2.5* würde das gleiche Ergebnis liefern).

27.4 Eigene Unterfunktionen definieren

Häufig verwendete Funktionsblöcke können in eigene Unterfunktionen ausgelagert werden. Dies dient der Vereinfachung und Übersichtlichkeit des Hauptprogramms.

```
Function Name (parameters*)
 Function-BLOCK ; Berechnungen
 Name = value ; Rückgabewert
EndFunction
```

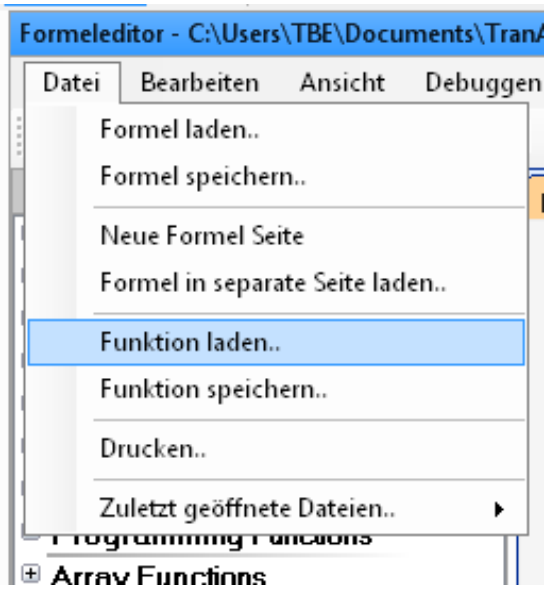
Der Rückgabewert wird als Formel mit dem Variablen-Namen, der dem Funktions-Namen entspricht, bestimmt.

Es können auch mehrere Unterfunktionen in einer Datei definiert werden.

Beispiel mit einer Datei mit dem Namen "MathOperators.fnc":

```
Function Addition(value0, value1)
 Addition = value0 + value1
EndFunction
```

Normalerweise werden solche Unterfunktionen in eigenen Dateien abgelegt. Die Namen dieser Dateien haben die Erweiterung *.fnc.
Im Menü ***"Datei" / "Funktion laden"*** bzw. ***"Funktion speichern"*** können diese Funktions-Dateien verwaltet werden:



Um eine Unterfunktion verwenden zu können, muss die Datei mit der Unterfunktion oben in den Hauptformeln deklariert werden. Dies wird mit dem **Schlüsselwort "*using*"** gemacht:

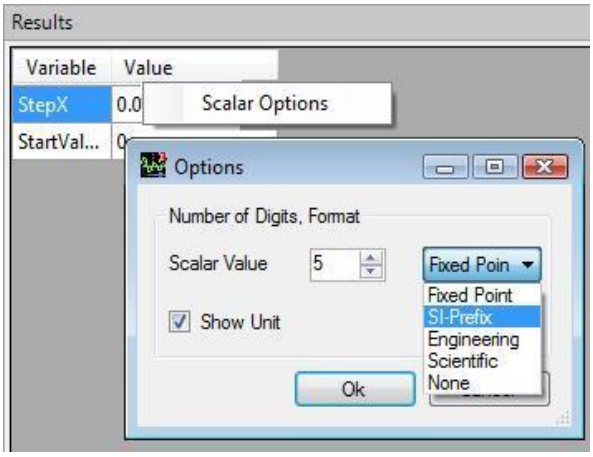
```
using "MathOperators.fnc"
result = Addition(2, 3)
```

Unterfunktionen können sich aber auch im gleichen Formel-Fenster unterhalb der Instruktion ***EndFormula*** befinden. Dies erleichtert vor allem die Arbeit bei der Entwicklung von eigenen Funktionen. In einem solchen Fall kann in den Haupt-Formeln die Anweisung ***using ()***" entfallen.
Unterfunktions-Aufrufe können sich auch in anderen Unterfunktionen befinden.

Eine Sammlung von Standard-Unterfunktionen wird mit TranAX mitgeliefert.
Damit sind einer Erweiterung der Berechnungen keine Grenzen gesetzt.
Der Anwender hat auch die Möglichkeit eine eigene Sammlung von Funktionen anzulegen.

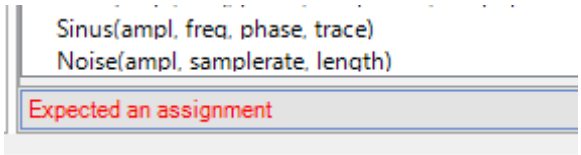
27.5 Zahlenformat der Skalar-Resultate

Die Darstellungsart der Skalar-Resultatwerte kann durch einen Rechtsklick auf die Zelle "Value" angepasst werden.



27.6 Fehlermeldungen

Allfällige Fehler in den Formeln, die durch den Parser festgestellt werden können, werden unten links im Formelfenster angezeigt. Mit einem **Doppelklick** auf einer allfälligen **Fehlermeldung** am unteren Ende des Formeleditors **springt der Text-Cursor an die fehlerhafte Stelle im Code**. Dies verkürzt die Suche nach der entsprechenden Fehlerquelle.



Oft liegt der Fehler in der unmittelbar davorliegenden Formel.

27.7 Übersicht der Funktions-Gruppen

Eine komplette Liste mit Funktionen ist im Anhang gelistet. Die verfügbaren mathematischen Funktionen sind in die folgenden Gruppen unterteilt:

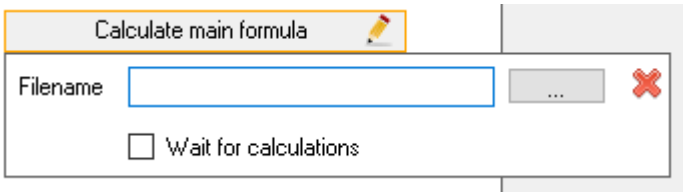
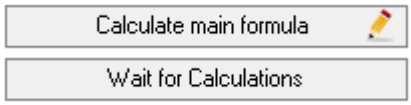
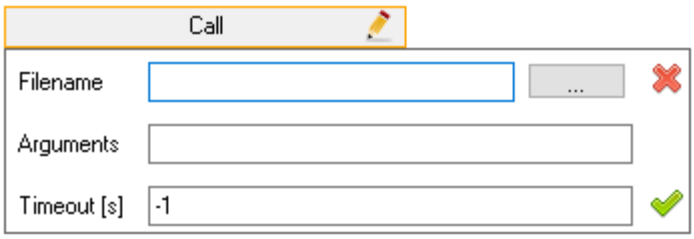
Name	Beschreibung
Channels	Hardware Signal-Quellen
All Functions	Alle verfügbaren Funktionen in alphabetischer Reihenfolge
Base Functions	Basis-Funktionen
File Functions	Operationen die sich auf Dateien beziehen
Signal Analysis	In der Regel Einzelwerte als Resultat
Signal Processing	In der Regel Kurven als Resultat
Filter Functions	Diverse Filter Funktionen
Programming Functions	Wenn/Dann-, Schlaufen usw.
Array Functions	Funktionen zum Arbeiten mit Arrays
Exponential and Trigonometric	Trigonometrische Funktionen
Spectrum (FFT)	Erzeugen und Berechnen von Spektren
Report Generator	Bilden und Ausfüllen von Berichten
Recording Parameters	Vornehmen von Hardware-Einstellungen
Layout Waveforms	Eigenschaften von Kurven-Anzeigen
Auto Sequence	Steuerung mittels Makro-Befehlen
Signal Generations	Funktionen, die Kurven generieren
Misc. Functions	Div. weitere Funktionen

28 Measurement Flow Control (MFC)

Die Messablaufsteuerung, Measurement Flow Control (MFC), ersetzt die bekannten Autosequenzen der Vorgängerversionen und erweitert diese um viele neuen Funktionen. MCF ist ein mächtiges Werkzeug, welches für die Messautomation verwendet werden kann.
Mit vordefinierten MFC Anwendungen können Messungen automatisiert und Daten direkt gespeichert werden.

In TranAX ist eine neuen Ribbon Bar mit dem Namen "Messablaufsteuerung" vorhanden.

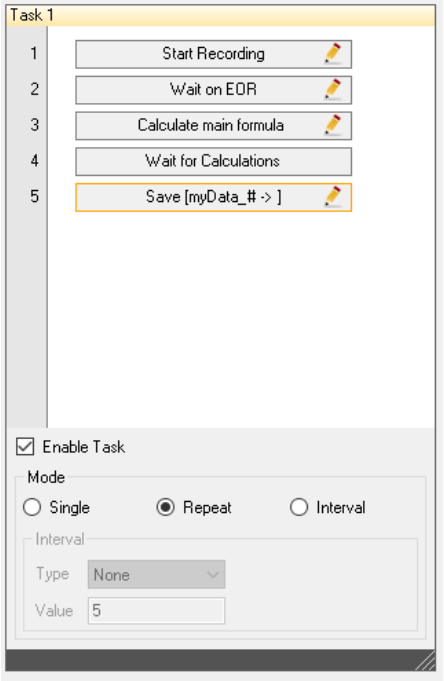
28.1 MFC Funktionen

Abort Sequence	Stopp den Messablauf des MFC. Alternativ kann die Taste "F8" oder der Stopp Knopf in der Ribbon-Bar gedrückt werden.
Autocalibration	Starte die Autokalibration des verbunden TraNET Gerätes.
Beep	Gibt den eingestellten Windows System Sound aus.
Calculate	Berechnet die Hauptformel im Formel Editor.  Optional kann auch eine vorhandene Formeldatei mit der Endung *.for ausgewählt werden  Wenn "Wait on calculation" aktiviert ist, wird gewartet bis die Berechnung der Formel abgeschlossen ist, bevor das nächste Kommando ausgeführt wird. "Calculate" in Kombination mit "Wait for Calculations" hat denselben Effekt.. 
Call	Startet eine Externe Anwendung oder eine Batchdatei.  File ist der Dateiname der Anwendung, die gestartet werden soll. Mit Arguments könne noch weitere Konsolen-Parameter und Argumente übergeben werden. Falls sich eine Anwendung nicht automatisch

	schliessen sollte, kann mit Timeout eine Zeit definiert werden bis das nächste Kommando ausgeführt wird.
Comment	Mit Comment können Kommentare ohne weitere Funktion in einen Task eingefügt werden. Informationen wie Erstellungsdatum oder Autor des Messablaufs können so einfach direkt hinterlegt werden. 
Delay	Wartet die angegebene Zeit in Sekunden. Es können ganze Zahlen und auch mit Nachkommastellen (Integer und Float) verwendet werden. Bei längeren Delays wir die verbleibende Zeit als Countdown in Sekunden dargestellt.
Disable External Start	TraNET Geräte besitzen einen Externe Start Eingang, verfügbar auf dem 25pol D-Sub Stecker des Sternverteilers. Diese Funktion verhindert den Externen Start. Siehe auch die Funktion "Enable External Start"
Enable External Start	Aktiviert den Externen Start Eingang. Siehe auch die Funktion "Disable External Start"
Event	Erzeugt ein Event, dieses kann dazu verwendet werden, weitere Task zu starten.
Exit Loop	Verlässt eine aktiven Loop, wird häufig in Kombination mit einer If-Else Funktion verwendet, z.B. um den Status einer Variable oder Formel zu prüfen.
Formula	Mit der Funktion Formula kann eine Formel direkt in einem Fenster eingegeben und dann ausgeführt werden. Siehe auch "Calculate
If-Else	Mit der IF-Else Funktion können Variablen und Werten aus Formeln geprüft und entschieden werden, wie weitergemacht werden soll. Alle Tasks laufen unabhängig von den Andren, daher müssen Variablem werte mit den Funktionen **WriteforNext** und **ReadofPrevious** übergeben werden.
Load Formula	Lädt eine Formel Datei in das Hauptfenster des Formel Editors.
Load Layout	Lädt ein gespeichertes Layout
Load Settings	Lädt gespeicherte Hardware Einstellungen.
Pause	Zeigt einen Informationsdialog, welcher vom Benutzer bestätig werden muss.
Print	Druckt eine Page oder eine Waveform an den eingestellten Standarddrucker (Kurven werden nach jeder Messung aktualisiert). Das Layout kann in der Druckervorschau eingestellt und definiert werden. Der Name der Page oder Waveform, welches gedruckt werden soll, kann im Dialog angegeben werden.

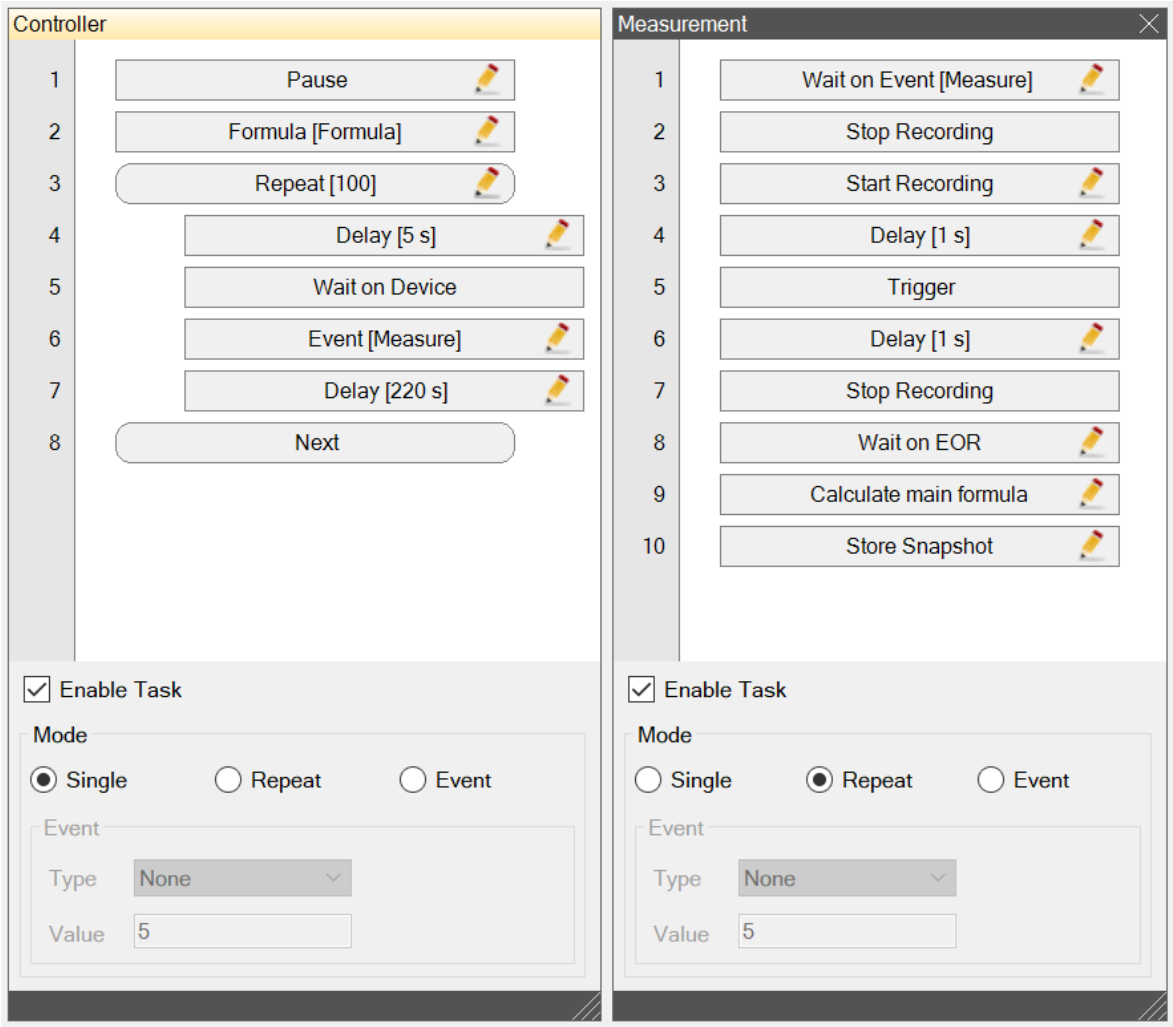
Quit Application	Schliesst TranAX. Falls Änderungen gemacht wurden und der Schreibschutz aktiv ist wird dies mit einem Dialogfenster gemeldet. Ohne Schreibschutz werden die Einstellungen gespeichert und TranAX gespeichert.
Repeat	Wird zusammen mit der Funktion Next verwendet. Die Anzahl der Wiederholungen kann definiert werden. Es ist auch möglich die Schlaufe mit unlimitierten Wiederholungen laufen zu lassen (Repeat Forever) Jeder Task bietet im unteren Bereich die Möglichkeit, ihn einmal (Single), immer wiederholend (Repeat Forever) oder in einem definierten Intervall auszuführen.
Save	Speichert eine oder mehrere aufgenommene Messkurve als eine *.tpc5 Datei.
Save spectrum	Speichert berechnete FFT Kurven in eine *.tps5 Datei.
Start Recording	Starte eine neue Messung, wird häufig in Kombination mit Wait on EOR verwendet. Wait on EOR kann auch im Dialog als Checkbox aktiviert werden. Die direkte Zuweisung der Messdaten zu einer Referenz kann ebenfalls direkt im Dialog vorgenommen werden.
Stop Recording	Stoppt eine laufende Aufnahme.
Store Page	Speichert eine komplette Page in eine *.tpd Datei.
Store Readout	Speichert den Inhalt einer Skalar Tabelle in eine Textdatei.
Store Snapshots	Speichert einen Screenshot einer Page oder Waveform. Das Dateiformat und die Auflösung kann in den TranAX Einstellungen definiert werden.
Trigger	Generiert ein Trigger Event.
Wait for Calucations	Wartet auf den Abschluss der Berechnungen der Hauptformel, Skalar Tabellen A und B und der Harmonischen. Wird in Kombination mit "Calculate" verwendet.
Wait on Armed	Wartet bis der Systemstatus des Systems "Armed" ist.
Wait on Device	Wartet bis das Geräte online ist, falls Netzwerkverbindung zu einem TraNET Device unterbrochen wurde.
Wait on EOR	Wartet bis die Aufnahme abgeschlossen ist (End Of Record, EOR), wird in Kombination mit "Start Recording" verwendet.
Wait on Event	Wartet auf ein Event, welches von einem anderen Task generiert wurde.
Wait on Trigger	Wartet bis das Signal getriggert wurde. Manuell oder ein Triggerevent.

28.2 MFC Beispiel



Einfaches Beispiel:
Messung starten, berechnen der Hauptformel im Formel Editor und schliesslich Speichern einiger Messkurven in einer Datei.

Beispiel mit zwei MFC-Tasks:
Der Controller prüft, ob das TraNET-Gerät bereit ist (eingeschaltet und angeschlossen), beginnt dann mit der Aufzeichnung, führt einige Berechnungen mit dem Formeleditor durch und speichert schließlich einen Schnappschuss der Wellenform.



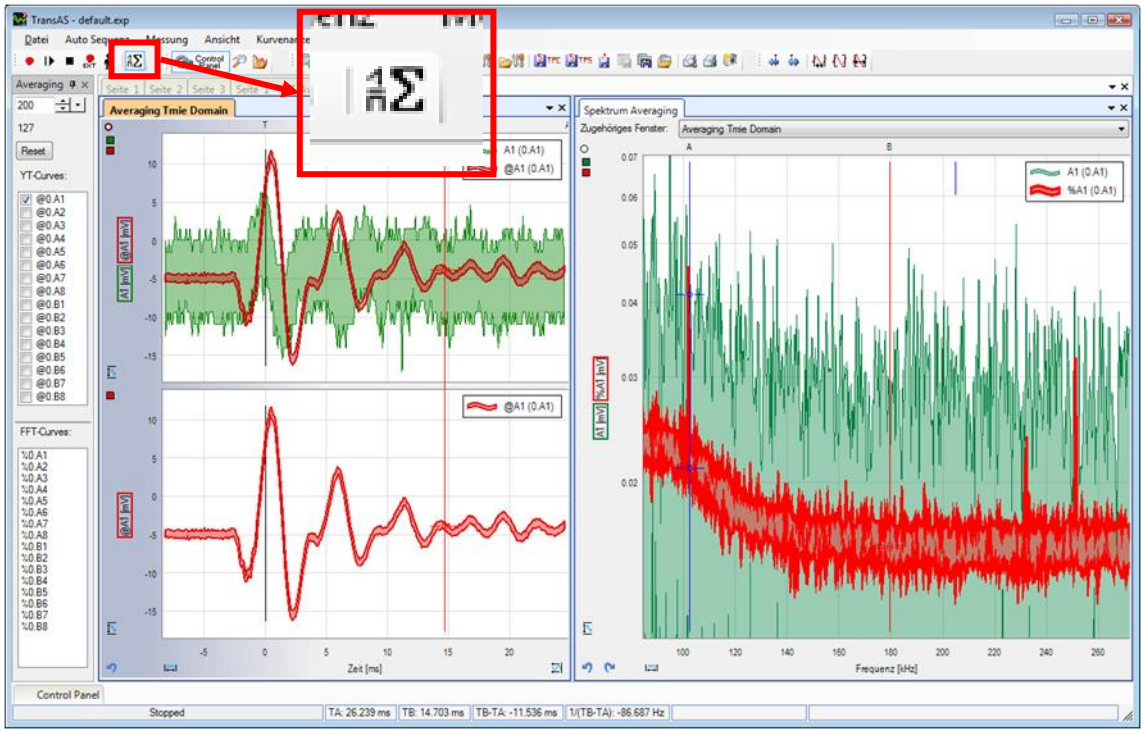
29 Mittelwertbildung über mehrere Aufnahmen

Die **Mittelwertbildung über mehrere Aufnahmen** ermöglicht, mehrere ähnliche, nacheinander aufgenommene Signale aufzusummieren und deren Mittelwert als Kurve darzustellen. Auf diese Weise können z.B. unregelmässige Störungen oder im Signal vorhandenes Rauschen vermindert werden.

Die Mittelwertbildung kann nur im Scope-Mode verwendet werden.

Um die Mittelwertbildung zu aktivieren, muss der Knopf **gedrückt sein**. Während die Mittelwertbildung aktiviert ist, ist auch das Averaging-Fenster geöffnet. Dieses wird mit Vorteil auf der linken Bildschirmseite angedockt.

29.1 Mittelwertbildung im Zeitbereich



Die untere Liste im Averaging-Fenster bezieht sich auf die Mittelwertbildung von Spektren.

Oben im Fenster ist ein Eingabefeld zu sehen, in welchem man die Anzahl der Aufnahmen eingibt, welche aufsummiert werden sollen. Die Zählanzeige darunter zeigt die Anzahl der bereits aufgenommenen (aufsummierten) Aufnahmen an. Wird die Limite erreicht, werden bei der nächsten Aufzeichnung die Summen gelöscht und eine neue Serie beginnt. Der Reset-Knopf dient dem vorzeitigen Zurücksetzen des Zählerstandes.

Durch Anklicken des Knopfs "Reset" wird vorerst nur intern festgehalten, dass bei der nächsten Aufnahme die Mittelwertbildung wieder bei eins beginnt. D.h. die aktuell vorliegenden gemittelten Kurven bleiben erhalten bis zum nächsten Aufnahme-Start.

In der Liste darunter sind alle Kanäle aufgelistet. Man kann individuell für jeden Kanal bestimmen, ob ein Mittelwert aufsummiert werden soll oder nicht.

Gleichzeitig dient diese Liste dazu, mittels Drag & Drop die gemittelten Kurven zur Darstellung zu bringen. Man kann mit der Maus einen Kanal ziehen und in einem Anzeige-Fenster fallen lassen.

Um die gemittelten Kanäle von den "normalen" Kanälen, d.h. von der zuletzt gemachten aktuellen Aufnahme unterscheiden zu können, haben die gemittelten Kanäle ein @ in ihrer Bezeichnung.

Nach jeder Aufnahme werden die zur Mittelwertbildung aktivierten Kanäle ausgelesen und zur aktuellen Summe hinzuaddiert. Dann werden alle Kurvenanzeigen neu dargestellt. So sieht man immer den aktuellen Mittelwert aus den bereits gemachten Aufnahmen. Wenn die Anzahl der Messungen die vorgegebene Limite erreicht, werden die Summen gelöscht und eine neue Serie beginnt.
Wenn im Control-Panel nicht "Single Shot" eingestellt ist (= Auto-Start), werden kontinuierlich Messungen gemacht und aufsummiert, bis die Limite erreicht ist. Dann wird angehalten. Durch Drücken des Start-Knopfs kann eine neue Serie gestartet werden.

Die aktuellen Mittelwert-Kurven können auch im Formel-Editor zu Berechnungen herangezogen werden, indem man sie mit **@** kennzeichnet, z.B. @c0A1 (anstatt c0A1) für Kanal 1.

Beispiel:

```
Save xx.tpc5, @0A1, @0A3-4
```

Um gemittelte Kurven in der Auto Sequenz zu speichern, muss die Installierte Software eventuell aktualisiert werden. Die folgenden Versionen werden als **Minimum vorausgesetzt**:

TranAX: **3.2.1.702** (Menü ***"Hilfe" / "Info"***)

29.2 Mittelwertbildung von Spektren

Auf die gleiche Art wie bei der Mittelwertbildung im Zeitbereich lassen sich durch **FFT ermittelte Spektren** aufsummieren und als Mittelwert darstellen, wobei nur von den Amplituden-Werten eine Mittelwertbildung durchgeführt wird. Die Phasenwerte werden nicht gemittelt.
Auch hier muss mit dem Knopf die Mittelwertbildung aktiviert sein.
Im unteren Teil des Average-Fensters sind alle Kanäle aufgelistet, welche für ein Mittelwert-Spektrum zur Verfügung stehen.
Ein "**%**" vor der Kanal-Nummer kennzeichnet, dass es sich um ein gemitteltes Spektrum handelt.
Es wird nur von denjenigen Signalen ein Mittelwert-Spektrum erzeugt, welche via Drag & Drop aus dieser Liste in ein geöffnetes Spektrum-Fenster gezogen wurden.

Aus dem Spektrum-Bild oben geht schön hervor, wie durch Mittelung von Spektren Frequenzen zum Vorschein kommen, welche sonst im Rauschen untergehen würden.

Falls bei Spektrum-Mittelung gewisse Spektrum-Parameter (z.B. FFT-Gewichtung oder Verstellen der Zeitmarken im zugehörigen Y-T-Fenster) geändert werden, erfolgt ein automatisches Zurücksetzen des Zählers.

Eine Mittelwertbildung braucht Rechenzeit und Speicherplatz, deshalb sollte man nur die benötigten Kanäle aktivieren. Auch sollte die Blocklänge möglichst kurz gewählt werden (vor allem beim Mitteln von Spektren).

30 Experimente und Experiment Sets

In TranAX bedeutet der Begriff "**Experiment**" eigentlich ein **Mess-Projekt**. Darin werden Einstellungen wie Messbereich, Abtastrate, Kanalnamen, die Anordnung der Fenster, Formeln, Autosequenzen usw. für einen bestimmten Messvorgang gespeichert.

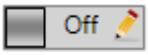
Experimente von TranAX 3 und TranAX 4 sind miteinander weitgehend kompatibel. Bestehende Experimente können weiterhin verwendet werden.
Werden alte Experimente von TranAX 3 geladen, so werden diese als Schreibgeschützes Experiment Set angelegt.

30.1 Dateistruktur eines Experiments

example.exp	Für jedes Experiment wird ein Unterverzeichnis angelegt. Dieses trägt in der Regel den Namen plus ".exp". Dies wird auch aus Gründen der rückwärts Kompatibilität weiterhin so gehandhabt.
data expr example.exp example.zip	Innerhalb dieses Experimentes sind die Unterverzeichnisse *data* und *expr*, darin werden die gemessenen resp. die berechneten Kurven abgelegt.
example.aut example.ctf example.lay example.tps.xml Settings.xml Snapshot.bmp	Ein Experiment Set ist eine Zip Datei, in der alle Einstellungen gespeichert sind, Layout, Hardware Settings, Auto Sequenz, Formeln und so weiter.
example.exp	In der Datei mit der Endung *.exp, welche den gleichen Namen hat wie das Experiment Verzeichnis selbst, steht der Verweis welches Experiment Set verwendet wurde.
Load last experiment after startup	Die Option "Letztes Experiment nach dem Starten laden" bietet die Möglichkeit, TranAX 4 im "Legacy Mode" zu verwenden, so dass bei jedem Start die Letzen Einstellungen geladen werden. Diese Einstellung kann auf der Startseite von TranAX vorgenommen werden.

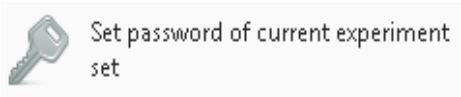
30.2 Schreibschutz für Experiment Sets

In der Regel werden alle Änderungen im Experiment beim Schliessen von TranAX in das aktuelle Experiment Set gespeichert. Für Anwendungen im Prüfbereich kann es von Vorteil sein, dass diese Einstellungen nicht mehr verändert werden.

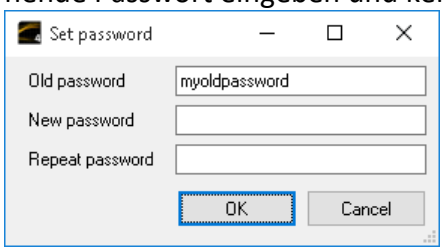
	Schreibschutz ist aktiv. Änderungen am Layout und Einstellungen können vorgenommen werden, diese werden aber beim Schliessen verworfen. Ein entsprechender Dialog warnt vor dem Verlust, so dass diese noch in einem anderen Experiment Set gespeichert werden können.
	Schreibschutz ist ausgeschalten. Beim Schliessen von TranAX werden alle Änderungen in das aktuelle Experiment Set gespeichert.

30.3 Passwortschutz für Experiment Sets

Weiter besteht die Möglichkeit dem Experiment Set ein Passwort zu vergeben. Ist dieses gesetzt, kann der Schreibschutz weiterhin aktiviert und ausgeschalten werden. Beim Ausschalten wird dann nach dem Passwort gefragt.



Um das gesetzte Passwort zu deaktivieren, kann der Dialog wieder geöffnet werden, das bestehende Passwort eingeben und kein neues Setzen.



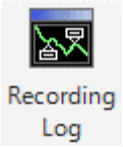
Sollte das Passwort nicht mehr bekannt sein, kann mit einem Texteditor in der Datei Settings.xml der Hash-Code des Passworts zwischen <Password> und </Password> gelöscht werden.

Die Funktionalität des Schreibschutzes und des Passwortschutzes dient rein der Verhinderung ungewollter Manipulationen, sei es aus Unwissenheit oder Unachtsamkeit.
Beide Funktionen bieten keinen Schutz gegen böswillige und Absichtliche Manipulationsversuche!

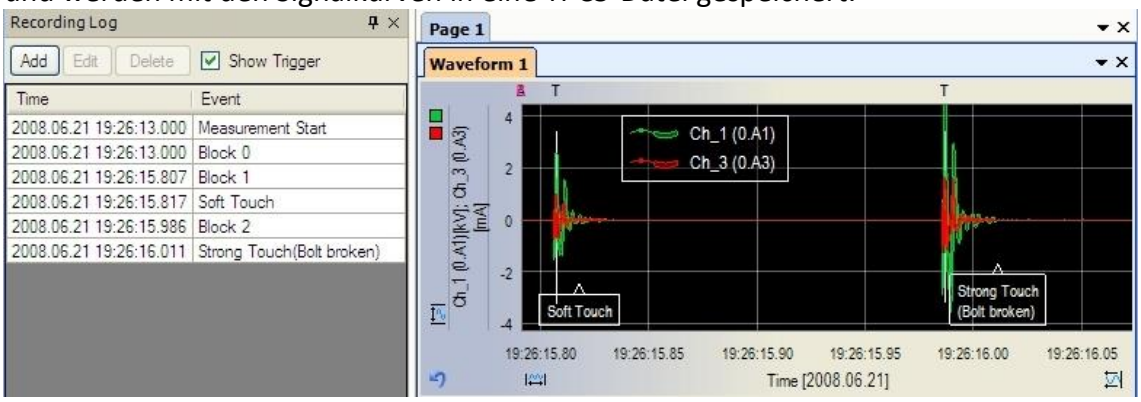
31 Diverse Controls

31.1 Aufnahmeprotokoll

Im Ribbon Tab "Layout" bei der Gruppe "Diverse Controls" den Knopf "Aufnahmeprotokoll" klicken und dass Aufnahmeprotokoll Fenster öffnet sich.



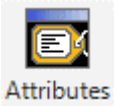
Das Aufnahmeprotokoll ist geeignet, um jederzeit Bemerkungen zu den Kurven-Daten während einer Aufzeichnung im Kontinuierlichen- oder ECR-Mode hinzuzufügen. Diese werden automatisch mit einem Zeitstempel versehen und können auf dem Bildschirm am entsprechenden Ort auf der Zeitachse angezeigt werden. Sie können auch im Nachhinein ergänzt oder editiert werden und werden mit den Signalkurven in eine TPC5-Datei gespeichert.

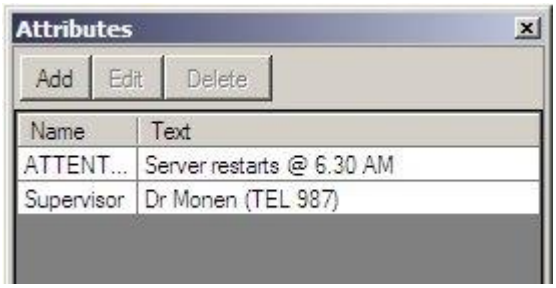


Falls Sie **Trigger zeigen** aktivieren, werden alle Trigger-Ereignisse inkl. Zeitstempel und entsprechende Kanäle aufgelistet. Sie können auch Ihre eigenen Kommentare mit der Schaltfläche **Hinzufügen** eingeben.

31.2 Attribute

Im Ribbon Tab "Layout" bei der Gruppe "Diverse Controls" den Knopf "Attribute" klicken und dass Attribute Fenster öffnet sich.



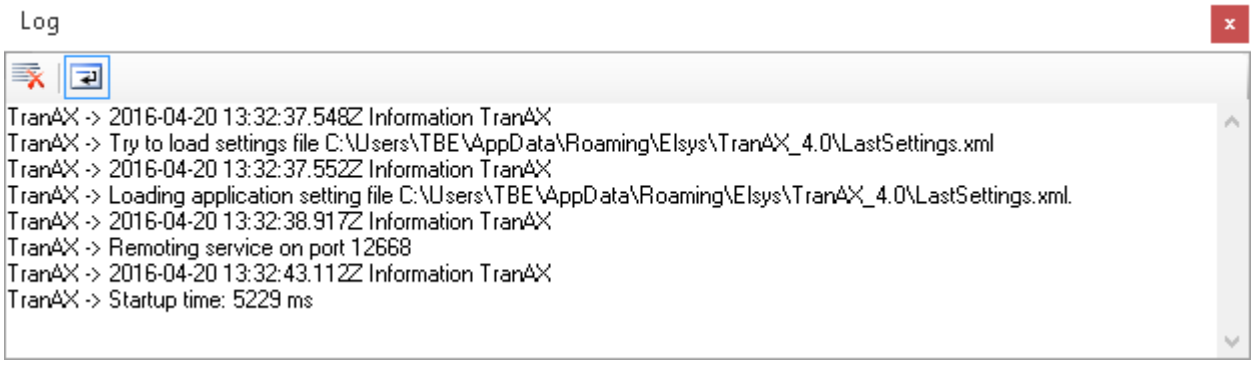


Unter ***"Attribute"*** können, im gleichen Sinn wie unter Aufnahmeprotokoll, **Text-Einträge** gemacht werden. Diese werden dann in den TPC5-Kurvendateien gespeichert. Attribut-Einträge haben jedoch im Unterschied zu Aufnahmeprotokollen keinen Zeitstempel. Attribute können Einträge wie Projektbezeichnung, Versuchsteilnehmer oder Hinweise zum Messaufbau enthalten.

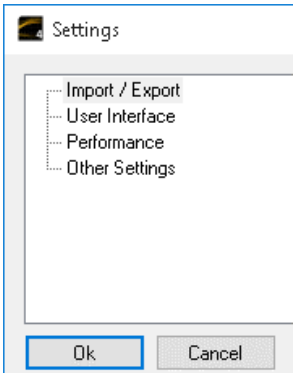
31.3 Fehlerprotokoll

Im Ribbon Tab "Layout" bei der Gruppe "Diverse Controls" den Knopf "Fehlerprotokoll" klicken und dass Fehlerprotokoll öffnet sich.

Das Fenster **Fehlerprotokoll** zeigt alle aufgetretenen Fehler. Die Fehlerliste kann nützliche Hinweise bieten, wenn sich TranAX suboptimal verhält.

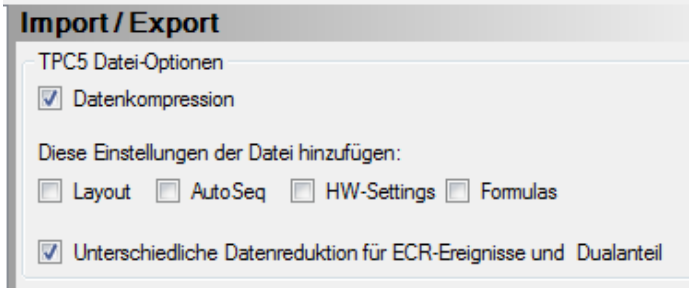


32 Einstellungen



Auf der Startseite mit der Schaltfläche "Einstellungen" und im Menu "Datei / Einstellungen" lässt sich der Dialog für die allgemeinen Einstellungen von TranAX öffnen.

32.1 Import/Export



Die hier vorgenommenen **Einstellungen** werden als **Standardwerte** in den Anderen Dialogen übernommen. Die Option **Datenkompression** wird anschliessend im **Speichern Dialog** als Standardeinstellung wieder angegeben.

Bei den TPC5 Datei-Optionen können die folgenden Einstellungen vorgenommen werden:

Datenkompression	Diese Einstellung bestimmt die Grundeinstellung des Fensters **Speichern.**
Einstellungen der Datei hinzufügen	Das Datenformat TCP5 bietet die Möglichkeit neben den Kurven-Daten auch Einstellungen in derselben Datei mit abzuspeichern
Layout	enthält alle Informationen der Bedieneroberfläche
Auto Seq	enthält die Informationen der Auto Sequenz
HW-Settings	enthält die Hardware-Einstellungen aus dem Control Panel
Formulas	enthält die Informationen des Formel Editors
Unterschiedliche Datenreduktion für ECR-Ereignisse und Dualanteil	Falls im ECR Datenerfassungs-Mode der Dual-Anteil mit aufgezeichnet wurde, kann hier freigeschaltet werden, dass ECR- und Dual-Anteil unterschiedlich reduziert werden können

Im ECR Datenerfassungs-Mode kann der langsamere Dual-Anteil mit aufgezeichnet werden. Das Eingabefeld **Reduktion für ECR-Events** ist nur sichtbar, wenn unter beim Ribbon Tab ***"Einstellungen" / "Einstellungen" / "Import /Export"*** die Einstellung "**Unterschiedliche Datenreduktion …**" aktiviert ist.

Für weitere Informationen zum Speichern von Signalkurven in das TPC5 Format, siehe Abschnitt Speichern.

32.2 User Interface (Benutzeroberfläche)

Hier könne Standardeinstellungen zur Benutzeroberfläche vorgenommen und definiert werden.

Language (•) English ( ) Deutsch	Wie im Menü ***"Extras" / "Sprache"*** kann auch hier die Sprache festgelegt werden. *Nach einem Wechsel ist ein Neustart von TranAX notwendig, um die Änderungen zu übernehmen.*
Toolbar ☐ Use Bigger Toolbar Icons	Die Icons für die Aufnahme-Kommandos können auch im Grossformat angezeigt werden
Mouse-Wheel ☐ Inverted Zooming ☐ Inverted Scrolling	Die Richtung, in der eine Kurve verschoben oder ein- und ausgezoomt wird kann mit Aktivieren des Invertierens umgekehrt werden.
Snapshot ( ) Bitmap DPI 96 ☐ Enable Constant Size (Pixel) (•) Vectorized (.emf) Width 1280 ☐ White Background Height 500	Die Kurvenanzeige wird mit Klick auf das Kamera-Icon in die Zwischenablage gelegt. Falls **Vektor** angewählt ist, so wird mit dem Bild der Kurvenanzeige eine Vektorgrafik im Enhanced Metafile-Format angelegt. Zudem kann die Grösse in Pixel, wie der Bildausschnitt danach erscheinen soll, gewählt werden. Wenn "***Weisser Hintergrund"*** markiert ist, wird der Hintergrund weiss gesetzt. Achtung: Helle Kurvenfarben sind dann z.T. schlecht sichtbar auf dem Bild. Daher sollten die Kurvenfarben eher dunkel gewählt werden.
Scalar Function Tables ☐ Colorful Channel Cells ☐ Show Readout Markers On Trace	Wenn "***Farbige Kanalzellen***" eingeschaltet ist, wird die Hintergrundfarbe von Resultat-Zellen in Skalar-Tabellen auf die Farbe der jeweiligen Signalkurve gesetzt. Wenn "***Markierungen auf Kurven..*** " eingeschaltet ist, werden Kurven nach entsprechenden Skalar-Berechnungen im zugehörigen Kurvenfenster mit Symbolen versehen. Es sind dies kleine Kreise bei Amplituden-Werten (z.B. Maximum, Peak-Peak usw.) oder kleine Rechtecke bei Zeit-Angaben (z.B. RMS-Periodic usw.).
Default Tab Window Placement Zoom Bottom XY Right Marker Bottom FFT Bottom Scalar Function Table A Right Scalar Function Table B Right Harmonics Table Right	Definiert die Position, an der ein neues Fenster angelegt wird.
Misc Grid Color Default Color ☐ Minor Grid Back Color ( ) Black ( ) White ☑ Show cursor and trigger letters in statusbar Text / Plot Label / Indicators Font	Via diesen Parameter kann die Gitter- und Hintergrundfarbe festgelegt werden. Mit "Minor Grid" kann auch das feinere Gitter zugeschaltet werden. Die Cursor Buchstaben (A, B, C ...) können oben innerhalb der Kurvenanzeige oder auf der Statuslinie darüber dargestellt werden.

Recording Log Position Bottom Font ☐ Recording Log vertically ☐ Center text ☐ Adapt to biggest	Einstellungen zum Recording Log.

32.3 Performance (Systemleistung)

Data Refresh Rate 50ms ☐ Indicate in Plot	Hier wird angegeben, in welchen Intervallen die Daten der Kurvenanzeige aktualisiert werden sollen. TranAX verwendet diese Information, um zu gewichten, wie die Systemleistung zwischen Datenvisualisierung und Datenauswertung erfolgen soll. Es gilt herauszufinden, welcher Wert für welche Anwendung geeignet ist. Wenn TranAX z.B. im kontinuierlichen Mode mit vielen Kanälen nur noch stark verzögert reagiert, sollte die **Daten Aktualisierungsrate** höhergestellt werden. Mit dem Knopf **Im Plot anzeigen** wird gewählt, ob diese Information in der Kurvenanzeige eingeblendet werden soll oder nicht.
Maximum Visible Blocks Recording Active 10 Recording Inactive 50 ☐ Indicate in Plot	Mit dieser Einstellung kann festgelegt werden, wie viele Datenblocks bei aktiver oder inaktiver Messung maximal dargestellt werden sollen. Diese Einstellung bezieht sich auf Daten, die im Block- oder ECR-Mode erfasst wurden. Mit dem Knopf **Im Plot anzeigen** wird gewählt, ob diese Information in der Kurvenanzeige eingeblendet werden soll oder nicht.
Crosshair position for Cursors ○ Screen Data (Pixel) ◉ Raw Data (Hardware / File)	Die Y-Cursor-Position wird entweder aus den vorverarbeiteten Daten entnommen, die für die Bildaufbereitung verwendet werden, oder aus den Rohdaten. Die Selektierung *Rohdaten* zwingt TranAX, neue Daten anzufordern und kann deshalb die Systemleistung beeinträchtigen.
FFT Maximum Blocklength [MS] 10	Begrenzt die maximale Blocklänge für FFT Analysen, um die Systemleistung nicht unnötig zu beeinträchtigen.

32.4 Other Settings (Andere Einstellungen)

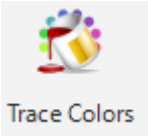
Bei einem ungewollten Beenden von TranAX (Stromausfall usw.) werden eventuell die Einstellungen nicht gespeichert. Alle vorgenommenen Änderungen seit dem letzten gewollten Speichern wären dabei verloren.

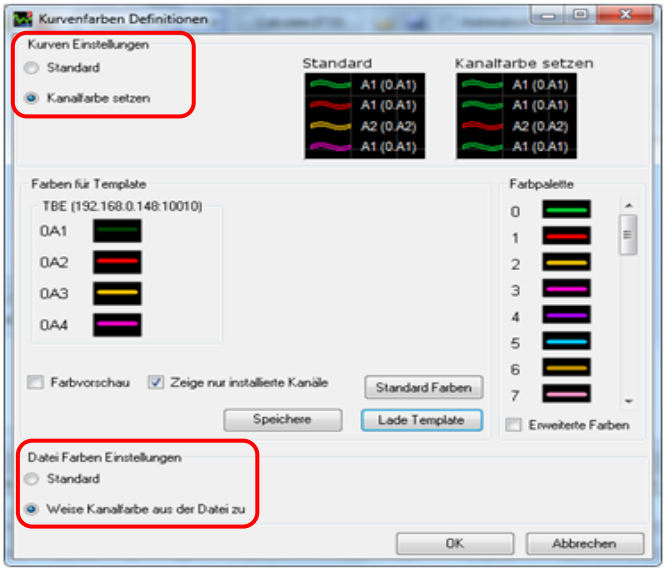
Auto Save (all settings) [min]
None

Im Menu Tab ***"Einstellungen"***, kann bei ***"Einstellungen"*** unter ***"Auto-Save"*** kann ein Zeitintervall vorgegeben werden, wonach die Einstellungen automatisch gespeichert werden. Das bestehende Experiment Set wird dabei überschrieben.

32.5 Kurven-Farben einem Kanal zuweisen

Es gibt die Möglichkeit, den **einzelnen Kanälen** eine Farbe zuzuweisen, welche dann immer für denselben Kanal verwendet wird. So kann man für sein spezifisches Experiment die Farben den Kanälen zuteilen. Im Ribbon Tab "Einstellungen, Gruppe "Einstellungen" beim Symbol "Kanalfarben" einen Dialog zur Verfügung, welcher dies ermöglicht.





Damit die **Farben benutzerspezifisch** zugeordnet werden können muss unter **"Kurven Einstellungen" "Kanalfarbe setzen" aktiviert** sein.
In der Gruppenbox **"Farben für Template"** hat man nun die Möglichkeit, **jedem Kanal** eine **Farbe zuzuweisen**. Die Farbe kann entweder über den "Color Chooser" (Farbe anklicken) oder via **Drag & Drop** aus der **Farbpalette rechts** gewählt werden. Bei der Farbpalette kann man die abgestuften Farben abrufen indem man "Erweiterte Farben“ aktiviert.

Hat man einmal seine Farbauswahl getroffen, kann man diese via **"Farbvorschau"** direkt in TranAX betrachten und beurteilen. Möchte man ein **Template** vorbereiten, welches die Farbzuweisung nicht bloss zu den Kanälen der installierten Module zulässt, kann man **"Zeige nur installierte Kanäle" deaktivieren**. Danach können die Farben der Kanäle von 8 Devices mit je 8 Modulen zu je 8 Kanälen definiert werden.

Man kann die getroffene Farbwahl als **Template** mit benutzerspezifischem Namen **speichern**. Mit "OK" werden die gewählten Farben definitiv zugewiesen und die Maske geschlossen. Beim Verlassen des aktuellen Experiments oder beim Schliessen von TranAX wird automatisch die Datei "**Lastcolors.ctf"** angelegt. Beim Erzeugen eines **neuen Experiments** wird automatisch die **aktuelle Farbauswahl übernommen**.

Einstellungen zu Kurvenfarben:

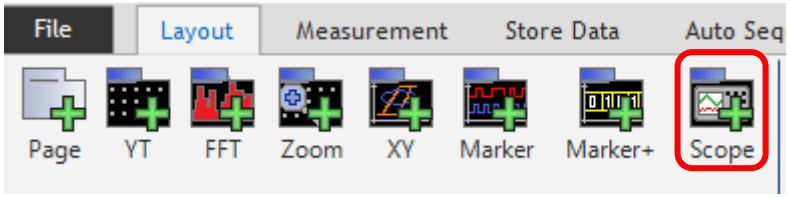
Standard	Jede Kurve erhält eine Farbe, welche von TranAX aufgrund der **Reihenfolge in den Kurvenfenstern automatisch** gewählt wird.
Setze Kanalfarbe	Jeder Kanal erhält die **selbstdefinierte Farbe.**

Datei Farben Einstellungen:

Standard	Jede Kurve erhält eine Farbe, welche von TranAX aufgrund der **Reihenfolge in den Kurvenfenstern automatisch** gewählt wird.
Weise Kanalfarbe aus der Datei zu	Kurven aus einer TPC5-Datei erhalten die Farben die **ursprünglich beim Speichern gewählt waren**.

33 SCOPE (Oszilloskop)

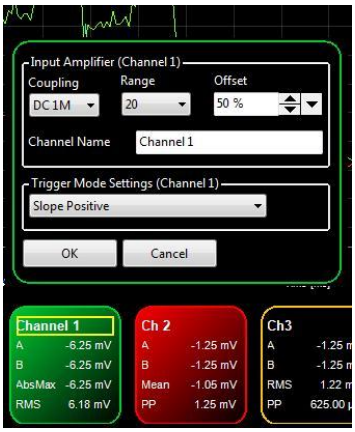
Mit SCOPE wird das Arbeiten bei einfacheren Anwendungen wesentlich erleichtert. TranAX verhält sich dabei wie ein Oszilloskop. Zwar immer noch ohne Drehknöpfe, aber für die Bedienung wurden selten gebrauchte Modes und deren Einstellungen stark in den Hintergrund gebracht.
Alle Manipulationen, welche die Kurven, deren Fenster, Achsen, Beschriftungen usw. betreffen verhalten sich gleich, wie bei einem normale Y/T-Fenster. Durch Klicken auf das Scope-Icon wird ein SCOPE-Fenster erzeugt.



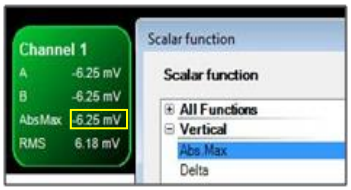
33.1 Kanal-Einstellungen

Im unteren Teil befindet sich für jeden Hardware-Kanal eine Bedienfläche.
Je nach Hardware sind 4 oder 8 solcher Bedienflächen vorhanden (bei Systemen mit mehr als 8 Kanälen, werden nur die ersten acht berücksichtigt).
Links befindet sich die Bedienfläche für den ersten Kanal (BNC A1). Die Bedienflächen sind vor allem durch die Namen der Kanäle gekennzeichnet.

Normalerweise wird ein gelber Rahmen um die ganze Bedienfläche gezeichnet, wenn sich der Maus-Cursor innerhalb der Fläche befindet. Durch Drücken der linken Maus-Taste kann der Kanal (inkl. Kurve) aktiviert oder deaktiviert werden. Durch das Aktivieren wird normalerweise die Kurve sichtbar.



Wenn sich der Maus-Cursor über einem Kanal-Namen befindet, wird dieser mit einem gelben Rahmen versehen. Durch Links-Klick öffnet sich eine Maske zur Einstellung der wichtigsten Kanal-Parameter.



Ein gelber Rahmen wird auch gezeichnet, wenn der Maus-Cursor über einem Skalarwerte-Bereich platziert wird. Durch Klick mit der linken Maus-Taste öffnet sich die Maske für die Auswahl und Spezifizierung einer Skalar-Funktion.

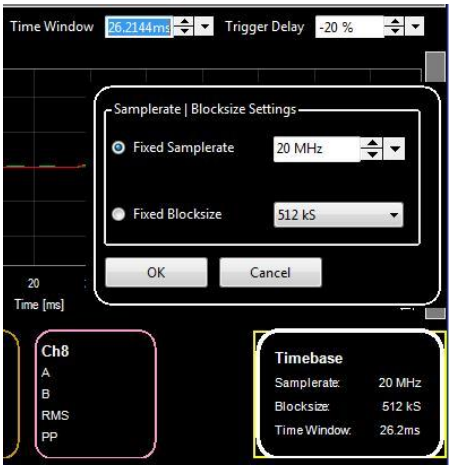
33.2 Knöpfe für Aufnahme-Befehle

Im oberen Bereich befinden sich die Befehlsknöpfe. Wie erwähnt kann damit wie mit einem Oszilloskop gearbeitet werden. Weitergehende Aufnahme-Modes (z.B. Multi-Block, ECR usw.) werden nicht unmittelbar ermöglicht.

Auto: Mehrfachauslösung mit "Auto-Trigger" (freilaufendes Oszilloskop).
Normal: Mehrfachauslösung, wartet auf Signal-Trigger und startet neu nach Abschluss der Aufnahme.
Single: Startet Aufnahme und wartet auf Trigger.
Nach Abschluss der Aufnahme wird gewartet.
Trigger : Das Programm schickt einen Trigger zur Hardware. Damit kann bei Fehlen eines Signal-Triggers eine laufende Aufnahme richtig abgeschlossen werden.
Stop: Eine laufende Aufnahme wird abgebrochen. Danach wird gewartet.

33.3 Zeit-Einstellungen

Mit dem **Time Window** Parameter kann der Zeit-Bereich eingestellt werden. Nach einer Aufnahme entspricht dies normalerweise dem Bereich bei voller X-Achse (**Knopf** **links unten**).



Der Zeit-Bereich errechnet sich wie folgt:

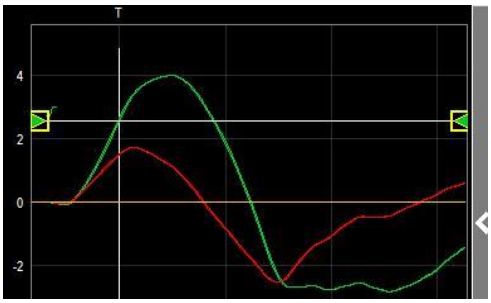
*T = Blocklänge * (1/Abtastfrequenz)*.

Der Anwender hat nun die Möglichkeit zu wählen, welche der beiden Termen vorgegeben und dann konstant sein soll. Durch Klicken auf dem **Timebase**-Feld rechts unten öffnet sich eine Maske. Darin kann zwischen "fixer Abtast-Frequenz" oder "fixer Block-Länge" und dem dazugehörenden Wert gewählt werden. Der andere (nicht fixierte) Wert wird danach automatisch aus dem unter **Time Window** gewählten Zeit-Fenster ermittelt.

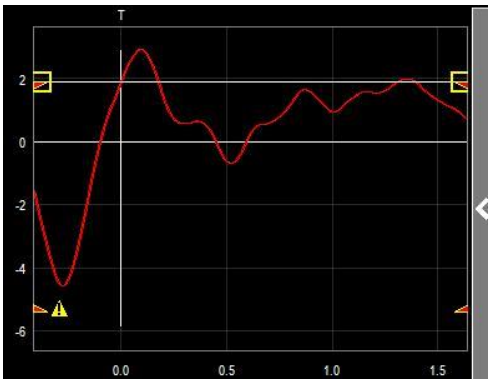
Unter **Trigger Delay** wird die Trigger-Verzögerung (-100% bis +200%) eingestellt. Diese Einstellung beeinflusst ebenfalls die Grenzen der Zeitachse. Allenfalls muss auch via
auf vollen Bereich der X-Achse gestellt werden.

33.4 Trigger-Bedingungen, Trigger-Pegel

Die Trigger-Bedingungen werden auch in der Maske der Kanal-Einstellungen gesetzt.
Sie sind bewusst einfach gehalten. D.h. es kann hier nur Flanken- oder Fenster-Trigger eingestellt werden.
Andere Einstellungen (inkl. Modes der Trigger-Option) könnten allenfalls direkt im Control-Panel gesetzt werden.

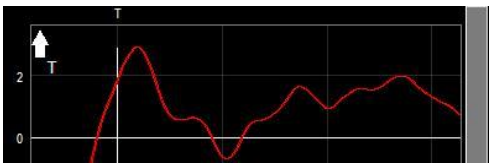


Die Pegel werden direkt im Kurven-Fenster eingestellt. Für jeden aktivierten Kanal, dessen Trigger-Mode nicht "Off" ist, erscheint am linken und rechten Fensterrand ein Dreieck-Symbol. Diese können mit der Maus gepackt und in der Höhe verschoben werden. Bei den Symbolen ist auch ersichtlich, ob auf die positive oder negative Flanke getriggert werden soll.

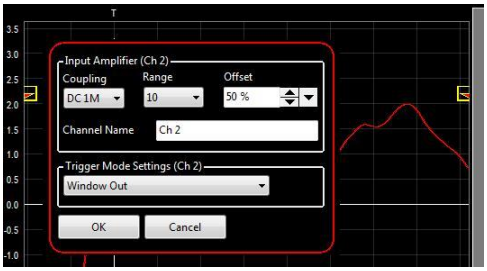


Fenster-Trigger werden durch zwei (vier) Halb-Dreiecke angezeigt. Diese können einzeln gepackt und verschoben werden.

Falls sich Pegel ausserhalb des Aufnahme-Bereichs befinden, wird ein Warndreieck angezeigt. D.h. es kann durch diesen Kanal kein Trigger ausgelöst werden.

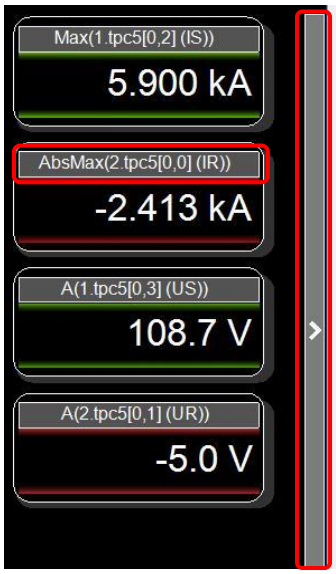


Falls sich die Pegel oben oder unten über dem Bildrand befinden (infolge Y-Zoom), wird dies durch weisse Pfeile angezeigt. Auch diese können mit der Maus gepackt und ins Fenster gezogen werden.



Wenn bei den Symbolen einfach mit der linken Maus-Taste geklickt (statt gepackt) wird, öffnet sich die Maske für die Kanal-Einstellungen (gleich, wie bei den Kanal-Namen in den Kanal-Bedienflächen).

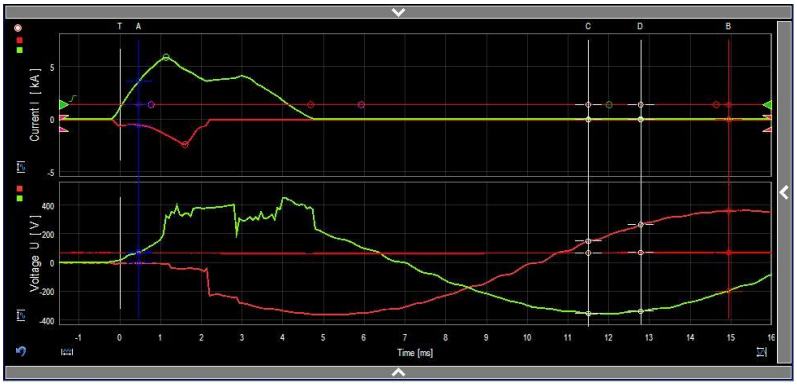
33.5 Digital-Anzeigen



Auf der rechten Seite können mehrere Anzeige-Panels dargestellt werden.
Diese lassen sich durch Klicken auf den vertikalen Balken rechts ein- oder ausblenden.
Die digitalen Wertanzeigen werden mit einer Skalar-Berechnung ermittelt. Grundsätzlich sind alle Berechnungen wie in den Skalar-Tabellen B möglich. Die Wertanzeigen sind überschrieben mit der Abkürzung des Resultats sowie dem Kanal-Namen. Wenn die Berechnung auf eine Kurve aus einer Datei angewendet wird, ist auch deren Namen eingeblendet. Mit einem Linksklick über einer solchen Überschrift kann die Maske zur Auswahl einer Skalar-Berechnung geöffnet werden.

33.6 Maximale Kurven-Darstellung

Durch einen Klick mit der linken Maus-Taste über dem oberen oder unteren Balken können die Bedienknöpfe bzw. die Kanal-Einstellungs-Flächen unterdrückt werden.



Dadurch wird der freiwerdende Bereich auch der Kurven-Anzeige zur Verfügung gestellt.

Anhang

34 Funktionsgruppen im Formeleditor

34.1 Array Functions

ArrayMultidimensional(number*)	Erstellt ein Mehrdimensionales Array, in den meisten praktischen Anwendungen ein 2D oder 3D Array.
	
	Example:
	`arr2D = ArrayMultidimensional(2, 2)`
	`arr2D(0, 0) = 0`
	`arr2D(0, 1) = 1`
	`arr2D(1, 0) = 1`
	`arr2D(1, 1) = 0`

Array(min to max) as dataType	Erstellt ein Array mit Type **dataType**. Der Parameter **dataType** kann vom Typ **double**, **string** or **complex** sein.
	min und **max** sollten ganzzahlig sein. Sie definieren die Grösse des Arrays. "0 to 3" ergibt vier Felder, "7 to 12" ergibt sechs Felder.
	
	Ein Array kann nur einen Datentyp beinhalten.
	
	Auf die einzelnen Werte kann mit Klammern **array(n)** zugegriffen werden. **n** muss dabei innerhalb der Werte liegen, in der das Array definiert wurde.
	
	Example:
	`arr1 = Array(0 to 3) as double`
	`arr2 = Array(0 to 3) as string`
	`arr3 = Array(0 to 3) as complex`
	`arr4 = Array(5, 6, 7, 12, 56)`
	`arr5 = Array("U1", "I1", "U2", "I2", "°C")`
	`arr6 = Array(5+0j, 3+45j, 12+23j, 78+45j, 34+12j)`
	
	`; return first item`
	`number = arr1(0)`
	
	`; set second item to "D"`
	`arr2(1) = "D"`
	
	`; set first item to complex number "5+2j"`
	`arr3(0) = 5+2j`

Array(min : max)	Erstellt ein Array mit der Länge **max - min + 1**. Alle Elemente in diesem Array werden mit dem Wert **min** + Index + 1 initialisiert.
	
	Example:
	`; arrExample = 1, 2, 3, 4, 5`
	`arrExample = Array(1:5)`

Complex	Schlüsselwort für Zahlenformat einer Variablen (komplexe Zahl), normalerweise zur Deklaration von Arrays. Diese Zahlen bestehen aus einem Real- und einem Imaginär-Teil **a+bj**.

	Man darf **j** oder **i** verwenden um eine Komplexe Zahl zu definieren.
	
	Example:
	`arrComplex = Array(0 to 3) as complex`
	`arrComplex(0) = 2 + 2j`
	`arrComplex(1) = -1 + 3j`
	`arrComplex(2) = -10.6 - 4.55j`
	`arrComplex(3) = 4j`
	
	`; = 1 + 5j`
	`complexRes1 = arrComplex(0) + arrComplex(1)`
	
	`; 18.2 - 42.4j`
	`complexRes2 = arrComplex(2) * arrComplex(3)`

ConvToArray(a)	Die Funktion **ConvToArray** konvertiert **a** zu einem Array. Wenn **a** eine Kurve ist gehen Zeit- und Trigger-Informationen dabei verloren.
	
	Example:
	`sig = c0A1`
	`arr = ConvToArray(sig)`

ConvToList(a)	Konvertiert **a** in eine Liste.
	
	Example:
	`ar$ = Array("A", "B", "C")`
	`CrsList = ConvToList(ar$)`

ConvToTrace(a, fs)	Die Funktion **ConvToTrace** konvertiert **a** in eine Kurve. Der Parameter **fs** ist die Abtastrate der neuen Kurve. **1/fs** ergibt den zeitlichen Abstand der einzelnen Samples in der neuen Kurve. Trigger Zeitpunkt wird bei 0s gesetzt, der Zeitstempel ist die aktuelle Uhrzeit des Rechners zur Zeit der Konvertierung.
	
	Example:
	`sf = 1E3 ;1kHz`
	`arr1 = array(1,2,3,4,5,6,7,8,9,10)`
	`trace = ConvToTrace(arr1, sf)`

Copy(a)	Formel Editor mit **Runtime 1.0** haben immer Kopien von List, Array oder Dictionary erstellt. Mit **Runtime 2.0** werden diese als Referenzen übergeben. Änderungen in der Kopie haben somit auch eine Auswirkung auf das Original. Dies ist auch bei Funktionsaufrufen mit einem Array, List oder Dictionary als Parameter der Fall. Möchten man nun explizit eine Kopie erstellen, so kann dies mit copy erstellt werden. Parameter **a** ist vom Typ List, Array oder Dictionary.
	
	Example:
	`; Select Runtime 2.0`
	`arrOrig = Array(0:9)  ; Creates an Array with entries  0 ... 9`
	`arrRef = arrOrig  ; Reference to`
	`arrCopy = Copy(arrOrig) ; Copy, no reference to arrOrig`
	
	`arrRef(4) = -5  ; Changes arrRef and arrOrig`
	`arrCopy(4) = 99  ; No impact to arrOrig`

Dictionary()	Erstellt ein Dictionary mit Wertepaaren (Key und Value). Key kann dazu verwendet werden, einen spezifischen Wert aus dem

	Dictionary zu lesen, ohne dessen genaue Positionen in der Liste zu kennen, im Gegensatz zu einem klassischen Array oder Liste. Siehe auch die Funktion **HasKey**.
	
	Example:
	`dictT = Dictionary()`
	`dictT("voltage") = 230`
	`dictT("current") = 10`
	
	`voltage = NotDefined`
	`current = NotDefined`
	
	`key = "voltage"`
	`if HasKey(dictT, key) = 1 then`
	`  voltage = dictT(key)`
	`endif`
	
	`key = "current"`
	`if HasKey(dictT, key) = 1 then`
	`  current = dictT(key)`
	`endif`

Double	Schlüsselwort für Zahlenformat einer Variablen (doppelte Genauigkeit, 64Bit), normalerweise zur Deklaration von Arrays. **Example:** `arr1 = Array(0 to 3) as double`

Flip(a)	Kehrt **a** vom ersten zum letzten Eintrag und gibt diese zurück. Diese Funktion kann unter anderem dazu verwendet werden, den Zeitversatz beim Filtern mit der Funktion **LowPass** zu eliminieren.
	
	Example:
	`trace = c0A1`
	
	`; creates a time offset`
	`trace = LowPass(trace, Bessel, 6, 10E3)`
	`trace = Flip(trace)`
	
	`; removes existing time offset`
	`trace = LowPass(trace, Bessel, 6, 10E3)`
	`trace = Flip(trace)`

GetArrayDimensions(array)	Gibt die Dimensionen eines Arrays zurück. Der Rückgabewert **variable** ist eine Zahl.
	
	Example:
	`; create two arrays`
	`arr = Array(0 to 9) as double`
	`arr3D = ArrayMultidimensional(4, 5, 7)`
	
	`; read dimensions of these arrays`
	`dimArr = GetArrayDimensions(arr) ; 1 dimensional`
	`dimArr3D = GetArrayDimensions(arr3D) ; 3 dimensional`

HasKey(dictionary, key)	Prüft die Existenz des Eintrages **key** in **dictionary**. Rückgabewerte sind: Existiert = 1, nicht gefunden = 0 Siehe auch Funktion **Dictionary**
	
	Example:
	`dictT = Dictionary()`
	`dictT("voltage") = 230`
	`dictT("current") = 10`

	`voltage = NotDefined`
	`current = NotDefined`
	
	`key = "voltage"`
	`if HasKey(dictT, key) = 1 then`
	`  voltage = dictT(key)`
	`endif`
	
	`key = "current"`
	`if HasKey(dictT, key) = 1 then`
	`  current = dictT(key)`
	`endif`

Insert(a, index, value)	Fügt die Variable **value** an der Position **index** in den String oder Liste **a**. ein. Der Wert **index** muss <= der Länge von **a** + 1 sein. **Example:** `text = Insert("Voltage", 7, "s") ; text= "Voltages"`

Length(a)	Gibt die Anzahl Elemente von **a** zurück.
	
	Example:
	`; String`
	`str = "Test"`
	`LenStr = Length(str) ; 4`
	
	`; Array`
	`arr = Array(0 : 7)`
	`LenArr = Length(arr) ; 8`
	
	`; Multi dimensional array`
	`arr3D = ArrayMultidimensional(4, 5, 7)`
	`x = Length(arr3D, 0) ; 4`
	`y = Length(arr3D, 1) ; 5`
	`z = Length(arr3D, 2) ; 7`

List()	**List** ist eine dynamische Datenstruktur. Alle Variablen-Typen (Zahlen, Strings, Kurven usw.) sind erlaubt. Auch gemischt. Sie kann unter Laufzeit erweitert oder gekürzt werden.
	
	Example:
	`mylist = List() ; List declaration`
	`mylist() = 134  ; Append a new elements`
	`mylist() = "Text"`
	`mylist() = c0A1`
	
	`; some calculation on trace element. Result is appended to list`
	`mylist() = Smooth(mylist(2),20)`
	
	`tr1 = mylist(2) ; get original trace`

Merge(a*)	**Merge** verbindet mehrere **a** zu einem. Dabei werden die einzelnen **a** nacheinander verkettet, die einzelnen Werte innerhalb der Arrays bleiben unverändert. **a** müssen vom selben Datentyp sein. Siehe auch die Funktion **MergeTraces**.
	
	Example:
	`arr1 = array(1, 2, 3, 4)`
	`arr2 = array(5, 6, 7, 8)`
	`arr3 = array(9, 10, 11, 12)`
	
	`arr_new = Merge(arr1, arr2, arr3)`
	`; arr_new contains now 12 elements.`

Remove(a [, index [, length]])
Remove(a, key)
	Entfernt ein Element aus der Liste, Dictionary oder String **a** at and der Position **index**. Der optionale Parameter **length** kann dazu verwendet werden, mehrere Einträge zu entfernen.
	
	Entfernt das Element **key** aus dem Dictionary **a**.
	
	Example:
	`; create an example`
	`dictT = Dictionary()`
	`dictT("test") = 10`
	`dictT("voltage") = 20`
	
	`; remove one item`
	`Remove(dictT, "test")`

String	Schlüsselwort für Textformat einer Variablen, normalerweise zur Deklaration von Arrays. Mit Strings können keine mathematischen Operationen durchgeführt werden, jedoch können Strings addiert (verkettet) werden ("123" + "abc", ergibt "123abc"). **Example:** `arr2 = Array(0 to 3) as string`

ZeroPadding(arr, NrOf-Zeros)	**ZeroPadding** fügt am Ende eines Arrays weitere Felder mit dem Wert 0 hinzu.
	arr ist ein Array, **NrOfZeros** die Anzahl zusätzlichen Felder im Array.
	
	Die bestehenden Felder werden nicht verändert, **neue sind auf 0 gesetzt**.
	
	Example:
	`; Array with 4 elements`
	`arr = array(1, 2, 3, 4)`
	`arr = ZeroPadding(arr, 10)`
	`; Now there are 14 elements in arr`

34.2 Auto Sequence Functions

AutoCalibration()	Führt eine Autokalibration aller Eingangskanäle durch.
	
	Example:
	`AutoCalibration()`
	`Pause("Autocalibration done!")`

Beep()	Der PC gibt einen Signalton von sich.
	
	Example:
	`for a = 0 to 10 step 1.0`
	`  delay(1)`
	`  Beep()`
	`next`

Call(cmd, args [, timeoutMs [, showWindow [, attachToProcess]])	Dieser Befehl startet ein anderes Programm. Dies kann entweder eine *.exe oder eine *.bat Datei sein. **timeoutMs** gibt an, wie lange auf die Beendigung des aufgerufenen Programms gewart werden soll. Wenn **timeoutMs** fehlt, wird sofort mit dem nächsten Formel-Befehl weitergefahren. Das aufgerufene Programm läuft dann als neuer Prozess

	parallel zu TranAX. Wenn **timeoutMs** grösser als Null angegeben wird, wartet die Autosequenz maximal so viele Sekunden auf die Beendigung des Programms. Wird das Programm vorher beendet, wird sofort mit dem nächsten Befehl weitergefahren. Ist **timeoutMs** negativ, z.B. -1, wird ohne zeitliche Beschränkung auf die Beendigung des Programms gewartet.
	
	cmd: Der Befehl welcher vom System ausgeführt wird.
	args: Zusätzliche Argumente für **cmd**
	timeoutMs: Die zu wartende Zeit (>0)
	showWindow: Fenster sichtbar oder versteckt
	attachToProcess: Fügt sich an den Hauptprozess an. Der Prozess wird beendet, wenn Hauptapplikation beendet wird.
	
	Example:
	`; Example executes a Python Script and waits until finished`
	`p = GetPath("data") ; Reference to "data" folder in Experiment`
	`pythonpath = "C:\Python\Python36\python.exe"`
	`arg = p + "\scan_sga.py"`
	
	`Call(pythonpath, arg, -1)`

DisableExternalStart()	Verhindert das Auslösen der Datenerfassung durch ein externes Signal am Start Record Eingang (normalerweise Pin 3 am 25pol. Stecker).
EnableExternalStart()	Ermöglicht das Auslösen der Datenerfassung durch ein externes Signal am Start Record Eingang (normalerweise Pin 3 am 25pol. Stecker).
GetTriggerStatus([clusterNr])	Gibt den Triggerstatus des Messystems zurück. Der optionale Parameter **clusterNr** kann dazu verwendet werden um einen spezifischen Cluster zu prüfen, falls mehrere definiert wurden. Rückgabewerte sind Disarmed = 0, Armed = 1, Triggered = 2

IsRecording()	Prüft ob eine Messung aktiv ist. Der Rückgabewert **variable** ist vom Typ Boolean. **Messung läuft = True**, ansonsten = False.
	
	Example:
	`if IsRecording()=1 then`
	`  ; do some debug stuff`
	`  else`
	`  pause("Not recording...")`
	`endif`

LoadLayout(filepath)	Lädt die Layout-Datei **filepath**.
LoadSettings(filepath)	Lädt die Einstellungen-Datei **filepath** ins Control Panel.
Pause([text [, variables])	Hält die Autosequenz an. Der Benutzer muss die Fortsetzung bestätigen. Dieser Befehl kann verwendet werden, um Meldungen wie z.B. "Bitte Prüfling aktivieren" auszugeben, oder dem Benutzer die Möglichkeit zu geben, eine abgeschlossene Messung anzuschauen, bevor die Autosequenz fortgeführt wird. Parameter **text** ist ein Text, mit **variables** können mehre Variablen übergeben und dann im Pause-Dialog angepasst werden. Rückgabewert **variable** ist ein Integer mit den folgenden Werten: 1= Abort 3= Continue **Example:**

	`items = 20`
	`text  = "This is a text."`
	`temperature = 23.5`
	
	`Pause("Please check!")`
	`val = Pause("Parameters", items, temperature, text)`

Print(name)	Druckt das Kurvenfenster **name**. (Die angezeigten Kurven werden nach jeder Messung automatisch aktualisiert). Der Ausdruck erfolgt so wie unter "Druckvorschau" spezifiziert.
	
	Example:
	`; print "Waveform 1" to the default printer`
	`Print("Page 1")`
	
	`; print "Page 1" to the default printer`
	`Print("Page 1")`

QuitApplication()	Die Hauptapplikation wird beendet.
SaveSpectrum(filepath, name, channels)	Speichert das Spektrum einer FFT-Kurvenanzeige in die Datei **filepath** (normalerweise ins Verzeichnis "data" des aktuellen Experiments). **name**: bezeichnet das FFT-Spektrum Fensters. Alle Kanäle **channels**, von welchen Spektren in die Datei gespeichert werden sollen. Kanäle bei welchen kein Spektrum gerechnet wurde (bzw. berechnet werden konnte) werden nicht gespeichert. Die Spektren werden genauso gespeichert, wie sie in dem Spektrum-Fenster aufgrund der Zeitfensterbegrenzungen in der ursprünglichen Zeitkurve berechnet wurden. Auch die FFT-Gewichtung kann nachträglich nicht geändert werden. Nur Spektren von Hardware-Kanälen (z.B. 0A1) werden akzeptiert. Bezüglich **filepath** gilt das gleiche, wie unter "File(....)" beschrieben. Wenn ein "#" am Ende des Dateinamens ist, wird dieses Zeichen durch eine fortlaufende Nummer ersetzt. **Example:** `SaveSpectrum("spectrum.tps5", "Spectrum 1", "0A1-4, 0B1, 0B3")`

StartRecording()	Startet eine Aufnahme.
	
	Die notwendigen Einstellungen müssen zuvor im **Control Panel** vorgenommen werden.
	
	Sie auch die Funktionen **Trigger**, **WaitonEOR** und **StopRecording**.
	
	Example:
	`; stop active measurements`
	`StopRecording()`
	
	`; start, trigger and wait until recording has stopped`
	`StartRecording()`
	`Delay(0.5)`
	`Trigger()`
	`WaitOnEOR()`
	
	`; save traces`
	`Save("myFilename#.tpc5", c0A1-c0A4)`

StopRecording()	Bricht eine laufende Aufnahme ab.
	
	Example:
	`; stop active measurements`
	`StopRecording()`
	
	`; start, trigger and wait until recording has stopped`
	`StartRecording()`
	`Delay(0.5)`
	`Trigger()`
	`WaitOnEOR()`
	
	`; save traces`
	`Save("myFilename#.tpc5", c0A1-c0A4)`

StorePage(filepath, name)	Speichert die Kurvenanzeige-Seite **page**in die Datei **filename** (normalerweise ins Verzeichnis "data" des aktuellen Experiments). Bezüglich **filepath** gilt das gleiche, wie unter "File()" beschrieben. Wenn ein "#" am Ende des Dateinamens ist, wird dieses Zeichen durch eine fortlaufende Nummer ersetzt. **Example:** `StorePage("mypage#.tdp", "Page 1")`
StoreSnapshot(filepath, name)	Speichert das Kurvenanzegefenster **name** in die Datei **filepath**. Es werden die Einstellungen unter "Extras / Einstellunge / Benutzeroberfläche ..." berücklsichtigt. Bezüglich **filepath** gilt das gleiche, wie unter "File(....)" beschrieben. Wenn ein "#" am Ende des Dateinamens ist, wird dieses Zeichen durch eine fortlaufende Nummer ersetzt.

Trigger()	Löst einen manuellen Trigger aus (sofern dies nicht bereits durch das Signal selber geschehen ist).
	
	Example:
	`; stop active measurements`
	`StopRecording()`
	
	`; start, trigger and wait until recording has stopped`
	`StartRecording()`
	`Delay(0.5)`
	`Trigger()`
	`WaitOnEOR()`
	
	`; save traces`
	`Save("myFilename#.tpc5", c0A1-c0A4)`

WaitForCalculations()	Die Operation **WaitForCalculations** wartet bis die Berechnungen von **Skalar-** und/oder **Harmonischen-Tabellen** abgeschlossen sind.
WaitForData([timeoutSeconds])	Wartet so lange bis mindestens ein gültiger Block aufgezeichnet wurde. Mit dem Parameter **timeoutSeconds** kann eine maximale Warte-Zeit (in Sekunden) vorgegeben werden. Ist dieser Parameter negativ oder fehlt er, wird unendlich lang gewartet. Im Continuous-Mode wird nicht gewartet. Das gleiche gilt, wenn im ECR-Mode die Dual-Aufzeichnung aktiviert ist. Man verwende dazu die Funktion **WaitOnEOR**. **Example:** `val = WaitForData()`

WaitOnEOR()	Wartet mit der weiteren Ausführung bis die Aufnahme beendet wurde (End Of Record). Falls im Control-Panel nicht Single Shot eingestellt ist, muss die Aufnahme manuell (oder durch den Befehl Stop Recording) abgebrochen werden um den EOR-Status zu erfüllen.
	
	Example:
	`; stop active measurements`
	`StopRecording()`
	
	`; start, trigger and wait until recording has stopped`
	`StartRecording()`
	`Delay(1)`
	`Trigger()`
	`WaitOnEOR()`
	
	`; Data available?`
	`if Length(c0A1) > 0 then`
	`  Save("myFilename#.tpc5", c0A1,c0A2,c0A3,c0A4)`
	`endif`

WatOnEOR()	Wartet mit der weiteren Ausführung bis die Aufnahme beendet wurde (End Of Record). Falls im Control-Panel nicht Single Shot eingestellt ist, muss die Aufnahme manuell (oder durch den Befehl Stop Recording) abgebrochen werden um den EOR-Status zu erfüllen.

34.3 Base Functions

a - b	Subtraktion der beiden Paramter **a** und **b**. Der zurückgegebene Wert **variable** ist im Allgemeinen vom selben Typ wie der Parameter **a**.
	
	Bitte beachten:
	
	• Strings können nicht subtrahiert werden.
	• Die Subtraktion einer Kurve mit einer Zahl ergibt einen entsprechenden Offset der Kurve.
	
	Example:
	`; Numbers`
	`n1 = 5-3 ; n1=2`
	
	`; Complex numbers`
	`n2 = -3+5j`
	`n3 = 5-10j`
	
	`n4 = n2 - n3 ; n2=-8+10j`
	`n5 = n2 - n1 ; n2=-5+5j`
	
	`; Strings cannot be subtracted!`
	
	`; Curves`
	`tr1[V] = Sinus(10, 50, 0, 1E3, 1E3) ; 10V amplitude (20Vpp), 50Hz`
	`tr2[V] = Sinus( 8, 30, 0, tr1)  ; 8V amplitude (16Vpp), 30Hz`
	`tr3[V] = tr1 - tr2 ; tr3 is a superposition of tr1 and tr2`
	`tr4[V] = tr3 - 5  ; tr4 with an offset of 5. The unit of the offset is the same as the physical unit of curves`
	
	`; Arrays`
	`; arr1 is an array of type double.`

<table>
<tr><td></td><td>

```
arr1 = array(0, 2, 5, 7)
arr2 = array(8, 7, -6, -5)

arr3 = arr1 - 1    ; arr3=(-1, 1, 4, 6)
arr4 = arr1 - arr2 ; arr4=(-8, -5, 11, 12)
```

</td></tr>
<tr><td>

a % b

</td><td>

Berechnet die **Modulo**-Operation (Euklidische Division), die Division der beiden Parameter **a** und **b**. Das ergibt einen Quotienten und den **ganzzahlingen Rest** als Ergebnis vom Rückgabewert **variable**.

Der zurückgegebene Wert **variable** ist im Allgemeinen vom selben Typ wie der Parameter **a**.

Example:

```
; Numbers
n1 = 5 % 3       ; n1=2
n2 = 10.3 % 2.1 ; n2=1.9

; Curves
tr1[V] = Sinus(10, 50, 0, 1E3, 1E3) ; 10V amplitude (20Vpp),
50Hz
tr2[V] = Sinus( 8, 30, 0, tr1)      ; 8V amplitude (16Vpp),
30Hz
tr3 = tr1 % tr2   ; sample for sample modulo of the two traces
tr4 = tr1 % 6     ; cuts off all values higher than 6

; Strings cannot be used with modulo!

; Arrays
; arr1 is an array of type double.
arr1 = array(0, 2, 5, 7)
arr2 = array(8, 7, -6, -5)

arr3 = arr1 % 4    ; arr3=(0, 2, 1, 3)
arr4 = arr1 % arr2 ; arr4=(0, 2, 5, 2)


; Modulo by 2 can also be used to determinate a even or odd
number
for a = 0 to 4 step 1.0
  str = "The number " + a + " is "
  if a % 2 = 0 then
    str +="even"
  else
    str += "odd"
  endif
     Pause(str)
next
```

</td></tr>
<tr><td>

a * b

</td><td>

Multiplikation der Parameter **a** und **b**. Der zurückgegebene Wert **variable** ist im Allgemeinen vom selben Typ wie der Parameter **a**.

Bitte beachten:

- Strings können nicht multipliziert werden.
- Die Multiplikation einer Kurve mit einer Zahl liefert eine Kurve mit entsprechender Skalierung.

Example:

```
; Numbers
n1 = 5*3 ; n1=15

; Complex numbers
```

</td></tr>
</table>

```
n2 = -3+5j
n3 = 5-10j

n4 = n2 * n3 ; n2=35+55j
n5 = n2 * n1 ; n2=-45+75j

; Strings cannot be multiplied!

; Curves
tr1[V] = Sinus(10, 50, 0, 1E3, 1E3) ; 10V amplitude (20Vpp),
50Hz
tr2[V] = Sinus( 8, 30, 0, tr1)      ; 8V amplitude (16Vpp),
30Hz
tr3[V] = tr1 * tr2 ; tr3 is a modulation of tr1 and tr2
tr4[V] = tr3 * 5   ; tr4 with scaling of factor 5

; Arrays
; arr1 is an array of type double.
arr1 = array(0, 2, 5, 7)
arr2 = array(8, 7, -6, -5)

arr3 = arr1 * 3    ; arr3=(0, 6, 15, 21)
arr4 = arr1 * arr2 ; arr4=(0, 14, -30, -35)
```

a / b

Division der Parameter Divisor **a** und Dividend **b**. Quotient = Divisor / Dividend Der zurückgegebene Quotient **variable** hat im Allgemeinen den gleichen Typ (nicht die gleiche Einheit) wie der Parameter **a**.

Bitte beachten Sie:

- Strings können nicht geteilt werden
- Die Division einer Kurve durch eine Zahl liefert eine Kurve mit entsprechender Skalierung.

Example:

```
; Numbers
n1 = 5/3 ; n1=1.666...

; Complex numbers
n2 = -3+5j
n3 = 5-10j

n4 = n2 / n3 ; n2=-0.52-0.04j
n5 = n2 / 5  ; n2=-0.6+1j

; Strings cannot be divided!

; Curves
tr1[V] = Sinus(10, 50, 0, 1E3, 1E3) ; 10V amplitude (20Vpp),
50Hz
tr2[V] = Sinus( 8, 30, 0, tr1)      ; 8V amplitude (16Vpp),
30Hz
tr3[V] = tr1 / tr2 ; tr3 is a division of tr1 with tr2
tr4[V] = tr1 / 5   ; tr1 with scaling of factor 1/5

; Arrays
; arr1 is an array of type double.
arr1 = array(0, 2, 5, 7)
arr2 = array(8, 7, -6, -5)

arr3 = arr1 / 3    ; arr3=(0, 0.66667, 1.6667, 2.3333)
arr4 = arr1 / arr2 ; arr4=(0, 0.28571, -0.83333, -1.4)
```


<table>
<tr><td>a^b</td><td>Potenzierung der Parameter Basis a und Exponent b. Potenz = Basis ^ Exponent Die zurückgegebene Potenz variable ist im Allgemeinen vom gleichen Typ wie der Parameter a.

Bitte beachten:

• Strings können nicht potenziert werden
• Komplexe Zahlen können nicht als Exponent verwendet werden
• Die Potenzierung einer Kurve mit einer Zahl gibt eine Kurve mit exponentieller Skalierung zurück.
• Diese Funktion liefert nur ein definiertes Ergebnis wenn die Parameter den Potenzierungsregeln entsprechen. Z.B. -3 ^ 0.5 ist nicht für reelle Zahlen definiert.

Example:
<pre>; Numbers
n1 = 5^3   ; n1 = 125
n2 = (-3) ^ 0.5 ; n2 = 0+1.7321j
n3 = 64 ^ (1/2) ; n3 = 8 (scuare root)
n4 = 64 ^ (1/3) ; n3 = 4 (cube root)

; Complex numbers
n5 = (-3+5j)^2 ; n5 = -16 -30j

; Strings cannot be exponentiated!

; Curves
tr1[V] = Sinus(10, 50, 0, 1E3, 1E3) ; 10V amplitude (20Vpp),
50Hz
tr2[V] = Sinus(8, 30, 0, tr1)   ; 8V amplitude (16Vpp),
30Hz
tr3[V] = tr1 ^ tr2 ; tr3 ist tr1 exponentiated with tr2
tr4[V] = tr1 ^ 5   ; tr1 ist tr2 with exponential scaling

; Arrays
; arr1 is an array of type double.
arr1 = array(0, 2, 5, 7)
arr2 = array(8, 7, -6, -5)

arr3 = arr1 ^ 3   ; arr3=(0, 8, 125, 343)
arr4 = arr1 ^ arr2 ; arr4=(0, 128, 6.4E-05, 5.9499E-05)</pre></td></tr>
<tr><td>a + b</td><td>Addition zweier Werte. a und b können eine Zahl, eine Kurve, ein String oder ein Array sein. Zahlen werden addiert und ergeben wieder eine Zahl. Kurven werden addiert und ergeben eine neue Kurve. Strings werden zusammen verkettet.

Die Addition einer Kurve mit einer Zahl ergibt einen entsprechenden Offset der Kurve.

Addition eines Strings und einer Zahl ergibt wieder einen String, wobei die Zahl als String interpretiert wird.

Bei der Addition eines Arrays mit einem Array werden die einzelnen Werte im Array miteinander addiert.</td></tr>
</table>

Die Addition eines Arrays mit einer Zahl ergibt einen Offset jedes einzelnen Wertes im Array.

Example:

```
; Numbers
n1 = 5+3 ; n1=8

; Complex numbers
n2 = -3+5j
n3 = 5-10j

n4 = n2 + n3 ; n2=5-5j
n5 = n2 + n1 ; n2=2+5j

; Strings
s1 = "c0A"+1     ; s1="c0A1" of type string
s2 = "c0A"+(1+1) ; s2="c0A2" of type string
s3 = "c0A"+1+1   ; s3="c0A11" of type string

; Curves
tr1[V] = Sinus(10, 50, 0, 1E3, 1E3) ; 10V amplitude (20Vpp),
50Hz
tr2[V] = Sinus( 8, 30, 0, tr1)      ; 8V amplitude (16Vpp),
30Hz
tr3[V] = tr1 + tr2 ; tr3 is a superposition of tr1 and tr2
tr4[V] = tr3 + 5   ; tr4 with an offset of 5. The unit of the
offset is the same as the physical unit of curves

; Arrays
; arr1 is an array of type double.
arr1 = array(0, 2, 5, 7)
arr2 = array(8, 7, -6, -5)

arr3 = arr1 + 1    ; arr3=(1, 3, 6, 8)
arr4 = arr1 + arr2 ; arr4=(8, 9, -1, 2)
```

Abs(a)

Gibt den absoluten (postiven) Wert von **a** zurück. Der zurückgegebene Wert **variable** ist vom selben Typ wie der Parameter **a**.

Kann auch in Kombination mit komplexen Zahlen verwendet werden, siehe auch die Funktionen **Real, Imag und Angle**.

Example:

```
; Numbers
n = -5
nAbs = Abs(n) ; nAbs = 5

; Complex numbers
c = -1j
cAbs = Abs(c) ; cAbs = 1

; Signals: Sine wave, 50Hz, 10Vpp
tr = Sinus(10,50,0,10E3,1E3)
trAbs = Abs(tr) ; Pulsating DC voltage, 100Hz
```

Diff(a)

Berechnet das Differential der Kurve **a**. Das Resultat **variable** ist wieder eine Kurve.

> Die Verwendung der Funktion **Diff** in Kombination mit hochfrequentem Rauschen kann zu unbefriedigenden Ergebnissen führen. Durch Filtern und Glätten wird dieser Effekt reduziert. Siehe auch die Funktionen **Lowpass** und **Smooth**.

Example:

	`; create a signal v[m/s] with 10[m/s^2] acceleration`
	`v[m/s] = Ramp(10,1E3,10E3)`
	`a[m/s^2] = diff(v)`

GetCrs(name [, areaIdx], cursor)	Gibt den **Zeitwert** in Sekunden oder den **Level** in der skalierten Physikalischen Einheit von **Cursor** zurück.
	
	name: Gültiges Waveform (Fensterbezeichnung) als String ("Waveform 1"). **cursor**: Aktiver und gültiger Cursor als String (z.B. "A" oder "AH").
	
	Bei mehreren Areas in einem Waveform kann der Parameter **AreaIndex** vom Typ Integer verwendet werden. Dieser startet bei 0, somit ist die zweite Area = 1.
	
	Horizontale Cursor, welche den Level darstellen, haben die Erweiterung "H". Somit liest man über Cursor "A" die Zeitinformation, mit Cursor "AH" den Level. Horizontale Cursor müssen explizit im Waveform aktiviert werden.
	
	Example:
	`; read from a waveform or first area in a waveform`
	`t1 = GetCrs("Waveform 1","A")  ; position of Cursor A`
	`l1 = GetCrs("Waveform 1", "AH") ; level of Cursor A Horizontal`
	
	`; read from second area in a Waveform`
	`t2 = GetCrs("Waveform 1", 1, "B")  ; position of Cursor B`
	`l2 = GetCrs("Waveform 1", 1, "BH") ; level of Cursor B Horizontal`

GetFFTTime-Window(name)	Gibt die **Positionen** der beiden Zeitmarker im YT Waveform zurück. Der Parameter **name** ist vom Typ String, der Rückgabewert **variable** ein zweidimensionales Array vom Typ Double. Dieser beinhaltet jeweils die Zeitinformationen des linken und rechten Zeitmarkers.
	
	Je nachdem, wie die Zeitmarker verschoben werden, kann die erste oder auch die zweite Zahl die Linke oder die Rechte Position darstellen. Gegebenfalls müssen diese beiden Zeiten für die weitere Bearbeitung noch sortiert werden.
	
	Siehe auch die Funktion **SetFFTTimeWindow**.
	
	Example:
	`timeMarker = GetFFTTimeWindow("Waveform 1")`
	
	`t1 = timeMarker(0)`
	`t2 = timeMarker(1)`
	
	`; make sure t1 is always smaller than t2`
	`if t1 > t2 then`
	`  tmp = t1`
	`  t1 = t2`
	`  t2 = tmp`
	`endif`
	
	`Pause ("Position of Time Window Indicator", t1, t2)`

Int(a)	Berechnet das Integral der Kurve **a**. Das Resultat **variable** ist wieder eine Kurve.
	
	Siehe auch die Funciton **Area** für numerische Resultate.
	
	Example:
	`; create a signal v[m/s] with 10[m/s^2] acceleration`
	`v[m/s] = Ramp(10,1E3,10E3)`
	`a[m/s^2] = diff(v)`
	
	`s_trace = Int(v)  ; creates a trace`
	`s_num  = Area(v)  ; numeric result`

Integer(a)	Rundet den Wert des Parameters **a** auf die nächste **ganze Zahl ab** (bis zu 15 Dezimalstellen).
	
	Der zurückgegebene Wert **variable** ist vom selben Typ wie der Parameter **a**. Diese Funktion wird vor allem zum Runden von Einzelwerten gebraucht.
	
	Example:
	`; Previous calculation`
	`; several periods of a signal`
	`Phas[°]=682.5`
	
	`; Phase should be +/-180°;`
	`Phas[°]=360*(Phas/360 - Integer(Phas/360+0.5))`
	`; Phas = -37.5°;`
	
	`n1 = Integer(0.5) ; n1=0`
	`n2 = Integer(0.9) ; n2=0`
	
	`; round according to rules`
	`for a = 0.4 to 1.6 step 0.1`
	`  valRound = Integer(a + 0.5)`
	`  s = "Value "+a+" is rounded "+valRound`
	`  Pause(s)`
	`next`

Limit(a, min, max)	Begrenzt **a** auf den Bereich **min** bis **max**.
	
	Der zurückgegebene Wert **variable** ist vom selben Typ wie der Parameter **a**.
	
	Diese Funktion kann verwendet werden, um eine allfällige Division durch Null (bzw. nahezu Null, bei Kurven mit Nulldurchgängen) zu vermeiden.
	
	Example:
	`; trace operation`
	`tr = Sinus(10, 50, 0, 1E3, 1E3) ; 10V amplitude -> 20Vpp`
	`trLimit = Limit(tr,-5, 8)  ; cut off signal`
	
	`; array operation`
	`arr = Array(0:20) ;create an array with the entries from 0 tp 20`
	`len = Length(arr) ; len = 21`
	
	`for a = 0 to len-1 step 1.0`
	`  arr(a) = Limit(arr(a), 2, 10) ; limit values of the array`
	`next`

MaxOf(a, b)	Diese Funktion liefert als Resultat **variable** die grössere von beiden Zahlen **a** und **b**.
	
	Example:
	`val1 = -12.3`
	`val2 = 20`
	
	`nlarger  = MaxOf(val1, val2) ; val = 20`
	`nsmaller = MinOf(val1, val2) ; val = -12.3`

MinOf(a, b)	Diese Funktion liefert als Resultat **variable** die kleinere von beiden Zahlen **a** und **b**.
	
	Example:
	`val1 = -12.3`
	`val2 = 20`
	
	`nlarger  = MaxOf(val1, val2) ; val = 20`
	`nsmaller = MinOf(val1, val2) ; val = -12.3`

SetCrs(name [, areaIdx], cursor, position)	Platziert den Cursor **cursor** auf die Position **position**, bei einem Zeitwert in Sekunden oder den Level in der skalierten Physikalischen Einheit.
	
	name ist ein gültiges Waveform (Fensterbezeichnung) als String ("Waveform 1"). **cursor** ist ein aktiver und gültiger Cursor als String (z.B. "A" oder "AH").
	
	Bei mehreren Areas in einem Waveform kann der Parameter **areaIndex** als Integer verwendet werden. Dieser startet bei 0, somit ist die zweite Area = 1 Horizontale Cursor, welche den Level darstellen, haben die Erweiterung "H". Somit liest man über Cursor "A" die Zeitinformation, mit Cursor "AH" den Level. Horizontale Cursor müssen explizit im Waveform aktiviert werden.
	
	Example:
	`; set in a waveform or first area in a waveform`
	`SetCrs("Waveform 1","A", 0.1)  ; position of Cursor A`
	`SetCrs("Waveform 1", "AH", 4.5) ; level of Cursor A Horizontal`
	
	`; set in the second area in a Waveform`
	`SetCrs("Waveform 1", 1, "B", -0.1)  ; position of Cursor B`
	`SetCrs("Waveform 1", 1, "BH", -2.1) ; level of Cursor B Horizontal`

SetFFTTime-Window(name, start, end)	Setzt die Positionen der beiden Zeitmarker im YT Waveform. Der Parameter **windowname** ist vom Typ String, **timeLeft** und **timeRight** sind vom Typ Double. Diese sind jeweils die Zeitpositionen der beiden Zeitmarker. Siehe auch die Funktion **GetFFT-TimeWindow**.
	
	Example:
	`tLeft = -1E-3 ; -1ms`
	`tRight = 1E-3 ;  1ms`
	
	`SetFFTTimeWindow("Waveform 1", tLeft, tRight)`

SetRange(a, b)
SetRange(a, min, max)
	Setzt den Y-Achsenbereich für **a** mit den Achseneinstellungen von **b**. Kann verwendet werden um Kurven mit demselben Y-Achsenbereich zu vergleichen.
	
	Jede Berechnung welche eine **Kurve** beinhaltet kann einen anderen Y-Achsenbereich haben. Darum kann man **SetRange** verwenden um bei allen Kurven denselben Y-Achsenbereich zu setzen für einen Vergleich. **Die Funktion hat keinen Einfluss auf die Samples**.
	
	Rückgabewert **variable** ist ein modifizierter Wert welcher den gleichen Typ wie **a** ist.
	
	Setzt den Y-Achsenbereich für **a** mit den Parametern **min** und **max**. Kann verwendet werden um Kurven mit demselben Y-Achsenbereich zu vergleichen.
	
	Jede Berechnung welche eine **Kurve** beinhaltet kann einen anderen Y-Achsenbereich haben. Darum kann man **SetRange** verwenden um bei allen Kurven denselben Y-Achsenbereich zu setzen für einen Vergleich. **Die Funktion hat keinen Einfluss auf die Samples**.
	
	Rückgabewert **variable** ist ein modifizierter Wert welcher den gleichen Typ wie **a** ist.
	
	Example:
	`trOrig1 = Sinus(10,50,0,1E3,1E3)  ; sine wave signal, 20Vpp`
	`trOrig2 = trOrig1 * 0.75  ; less amplitude`
	
	`trNew1 = SetRange(trOrig1, -15, 15) ; set range to +/-15V`
	`trNew2 = SetRange(trOrig2, trNew1)  ; copy range from previous trace`

Sqrt(a)	Berechnet die Quadratwurzel aus **a**.
	
	Example:
	`n1 = Sqrt(9)  ; n1= 3`
	`n2 = Sqrt(-9) ; n2= 0+3j`
	
	`; alternatively, Sqrt(a) = a^(1/2)`
	`n3 = 9 ^ 0.5`

34.4 Channels

EnChannels()	In Kombination mit **for each** kann **EnChannels** dazu verwendet werden, Daten von jedem Kanal zu lesen und zu analysieren. Es wird jeder verfügbare Messkanal im Messsystem zurückgegeben. Ausgeschaltete Kanäle werden ebenfalls gelistet.

	Example:
	`for each input in EnChannels()`
	`  maxVal = Max(input)`
	`  minVal = Min(input)`
	`next`

GetChannel(deviceIdx, boardIdx, inputIdx, blockNr)
GetChannel(name, blockNr)
	Gibt den Messblock **blockNr** vom spezifizierten Kanal **deviceIdx**, **boardIdx** und **inputIdx** zurück.
	
	Der Parameter **inputIdx** startet bei 1, **deviceIdx** und **boardIdx** starten bei 0.
	
	Mit der Funktion **NBlks()** kann die Anzahl erhältlicher Blöcke überprüft werden.
	
	Gibt den Messblock **blockNr** vom spezifizierten Kanal **name** zurück.
	Der Parameter **name** ist der Kanalname welcher benutzerspezifisch sein kann.
	
	Mit der Funktion **NBlks()** kann die Anzahl erhältlicher Blöcke überprüft werden.
	
	Example:
	`Blk = 0`
	`value = GetChannel(0, 0, 1, Blk) ; value = Device 0, c0A1, Blocks 0`

GetChannelDesc(trace)	Gibt ein Array mit drei Einträgen (length = 3) zurück. Diese beinhaltet die Werte für: **Device, Board and Channel**. Der Parameter **trace** definiert einen Hardware Kanal, Messkurven and Dateien können nicht verwendet werden!
	
	Die Parameter **Device** und **Board** beginnt mit 0 **Channel** beginnt mit 1.
	
	Example:
	`for each input in enChannels()`
	`  desc = GetChannelDesc(input)`
	`  range = GetRecordingPar(desc(0), desc(1), desc(2), 23)`
	`next`

GetChannelRanges(deviceIdx, boardIdx)	Gibt die verfügbaren Messbereiche der Karte **board** im Gerät **device** zurück. **Example:** `ranges = GetChannelRanges(0, 0); 1. Device, 1. Board, ranges= 0.1, 0.2, 0.5, 1, 2, 5, 10, 20, 50, 100`
GetChName(a)	Liefert den Namen von **a**. Dies können ein Hardwarekannal oder auch eine berechnete Kurve sein. **Example:** `trHw = c0A1`

```
name1 = GetChName(c0A1) ; hardware channel name
name2 = GetChName(trHw) ; hardware channel name

trCalc = Sinus(10,50,0,1E3,1E3)
name3 = GetChName(trCalc)   ; variable name
```

GetTrace (a[, index[, blockNr[, markerNr]]])

Gibt eine Kurve von **jeder verfügbaren Quelle in TranAX** zurück. Der Parameter **a** kann einer der folgenden Typen sein:

- Hardwarekanal (z.B. **c0A1**)
- Referenz (z.B. **ref1**)
- Datei (z.B. **"myFile.tpc5"**)
- Gemitteltes Zeitkurve (z.B. **"@0A1"**)
- Gemitteltes FFT-Signal (z. B. **"%0A1"**)

Der optionale Parameter **index** definiert den Kanal zum Lesen aus einer Datei, in jedem anderen Fall setzen Sie diesen auf 0. Der optionale Parameter **blockNr** definiert die Blocknummer, die mit 0 beginnt. Bitte beachten Sie, dass bei ECR-Aufzeichnungen der Dual-Anteil (niedrigere Abtastrate) Block 0 ist, falls aktiv.
Der letzte optionale Parameter **markerNr** ist entweder 1 oder 2, im Falle einer 14-Bit-Aufzeichnung können die oberen 2 Bits für externe digitale Signale verwendet werden, welche als Marker bezeichnet werden.

> Bitte beachten Sie, dass **Marker eine Hardwareoption** ist, welche in Ihrem TraNET-Gerät verbaut werden muss.

Example:

```
;Pause("Start Multi Block recording with 5 blocks. Click continue when done.")

; - Analyse hardware channel
; - Data from reference
; - Data from saved file

; Save channel A1 to a file
Save("myTrace.tpc5", "0A1") ; save all blocks
; Set channel A1 to Reference 1
SetReference(ref1,true, c0A1) ; true means references to the hardware channel


; 1. example
; read one block form the Multi Block recorded trace
; read 1st trace, 3rd block (begins with 0)
tr1 = GetTrace(c0A1, 0, 2)
tr2 = GetTrace("myTrace.tpc5", 0, 2)
tr3 = GetTrace(ref1, 0, 2)

; 2. example
; use for each loop
n1 = NBlks(c0A1)
n2 = NBlks(ref1)
n3 = NBlks(File("myTrace.tpc5",0))

cnt1 = 0
cnt2 = 0
cnt3 = 0

minVal1 = List()
```


	`minVal2 = List()`
	`minVal3 = List()`
	`get`
	`; Hardware channel`
	`for a = 0 to n1-1 step 1.0`
	`  tr1 = GetTrace(c0A1, 0, a)`
	`  minVal1() = Min(tr1)`
	`next`
	
	`; Reference`
	`for a = 0 to n1-1 step 1.0`
	`  tr2 = GetTrace(ref1, 0, a)`
	`  minVal2() = Min(tr2)`
	`next`
	
	`; From File`
	`for a = 0 to n1-1 step 1.0`
	`  tr3 = GetTrace("myTrace.tpc5", 0, a)`
	`  minVal3() = Min(tr3)`
	`next`

hardwarechannel	Stellt einen Hardware Kanal dar.

34.5 Enumerable Functions

EnBlocks(a, waitForBlocks)
EnBlocks(a, start, end)
EnBlocks(deviceIdx, boardIdx, inputIdx, waitForBlocks)
EnBlocks(deviceIdx, boardIdx, inputIdx, start, end)
	Die Funktion **enBlocks** liefert in Kombination mit for each loop die einzelnen Blöcke von **a**. Der Parameter **waitForBlocks** definiert ob die Formel wartet bis neue Blöcke vorhanden sind.
	
	Die Funktion **enBlocks** liefert in Kombination mit for each loop die einzelnen Blöcke von **a**. Die Parameter **start** und **end** welche Blöcke aus **a** gelesen werden.
	
	Die Funktion **enBlocks** liefert in Kombination mit for each loop die einzelnen Blöcke aus einem Hardwarekanal welche durch die Parameter **deviceIdx**, **boardIdx** und **inputIdx** definiert sind. Der Parameter **waitForBlocks** definiert ob die Formel wartet bis neue Blöcke vorhanden sind.
	
	Achtung: **deviceIdx** und **boardIdx** beginnen bei 0, **inputIdx** bei 1.
	
	Die Funktion **enBlocks** liefert in Kombination mit for each loop die einzelnen Blöcke aus einem Hardwarekanal welche durch die Parameter **deviceIdx**, **boardIdx** und **inputIdx** definiert sind. Die Parameter **start** und **end** welche Blöcke aus **a** gelesen werden.
	
	Achtung: **deviceIdx** und **boardIdx** beginnen bei 0, **inputIdx** bei 1.
	
	Example:
	`;multi block recording`
	`trace = c0A1`
	`minVal = List()`
	`for each item in  enBlocks(trace,false)`
	`  ; write min of each block into a list`
	`  minVal() = Min(item)`

	`next`

EnChannels()	In Kombination mit **for each** kann **EnChannels** dazu verwendet werden, Daten von jedem Kanal zu lesen und zu analysieren. Es wird jeder verfügbare Messkanal im Messsystem zurückgegeben.
	
	Ausgeschaltete Kanäle werden ebenfalls gelistet.
	
	Example:
	`for each input in EnChannels()`
	`  maxVal = Max(input)`
	`  minVal = Min(input)`
	`next`

EnEvents(a, type, start, end, level, hysteresis)	Die Funktion **enEvents** ist vergleichbar mit FindEvent, wird aber in Kombination mit einem for each loop verwendet. So können jeweils alle Ereignisse in **a** von **start** bis **end** gesucht werden. Der Parameter **type** definiert welche Flanke des Signals **a** gefunden werden soll. Die Parameter **level** und **hysteresis** definieren dann wo die Flanke genau gefunden werden soll. Rückgabewert **variable** ist dann die Position als Zeitwert in Sekunden.
	
	Example:
	`trace = c0A1`
	`; Slopes: 0=positive, 1=negative, 2=both`
	`type = 0`
	`t1 = TBegin(trace)`
	`t2 = tEnd(Trace)`
	`tEvents=list()`
	
	`for each item in enEvents(trace, type, t1, t2, 0, 0.1)`
	`  ; list with time stamps`
	`  tEvents() = item`
	`next`

EnSlices(a, size [, doUntilEOR])
EnSlices(a, size, start, end)
	enSlices liefert in Kombination mit **for each loop** einzelne Kurvenstücke der grösse **size** zurück. **a** ist eine Messkurve oder Messkanal. **doUntilEOR** vom Typ Boolean, ist dieser gleich True wird gewartet bis die gesamte Messung abgeschlossen ist. Bei False kann zur Laufzeit, z.b. im Continuous Mode einzelne Stücke gelesen und analysiert werden.
	
	enSlices liefert in Kombination mit **for each loop** einzelne Kurvenstücke der grösse **size** zurück. **a** ist eine Messkurve oder Messkanal. Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll.
	
	Example:
	`trace = c0A1`
	`for each item in enSlices(trace, 1E3)`
	`  ; your code here`
	`  tr = diff(item)`
	`next`
	
	`; continous calculation until measurement is done`
	`cntr = 0`
	`for each item in EnSlices(c0A1,1E3, true)`

	`cntr += 1`
	`next`
	`Pause("Numer of slices:"+cntr)`

34.6 Exponential and Trigonometric

Acos(a)	Berechnet den **Arkuscosinus** von **a**. Der Resultattyp entspricht wieder **a**.
	
	Werte die ausserhalb +/-1 liegen werden als nicht definiert gekennzeichnet.
	
	Ein Zahlen-Resultat wird in Grad (0° bis +180°), ein Kurven-Resultat in Radiant (0 bis Pi) angegeben.
	
	Example:
	`a1 = 1`
	`s1 = ACos(a1) ; 0°`
	
	`a2 = 0.5 ;`
	`s2 = ACos(a2) ; 60°`
	
	`; convert rad <-> deg`
	`; deg = 360 / (2*Pi) * rad`
	`; rad = (2*Pi) /360  * deg`

Asin(a)	Berechnet den **Arkussinus** von **a**. Der Resultattyp entspricht wieder **a**.
	
	Werte die ausserhalb +/-1 liegen werden als nicht definiert gekennzeichnet.
	
	Ein Zahlen-Resultat wird in Grad (-90° bis +90°), ein Kurven-Resultat in Radiant (-Pi/2 bis +Pi/2) angegeben.
	
	Example:
	`a1 = 1`
	`s1 = ASin(a1) ; 90°`
	
	`a2 = 0.5`
	`s2 = ASin(a2) ; 30°`
	
	`; convert rad <-> deg`
	`; deg = 360 / (2*Pi) * rad`
	`; rad = (2*Pi) /360  * deg`

Atan(a) **Atan(y, x)**	Berechnet den **Arkustangens** von **a**. Der Resultattyp entspricht wieder **a**. Ein Zahlen-Resultat wird in Grad (-90° bis +90°), ein Kurven-Resultat in Radiant (-Pi/2 bis +Pi/2) angegeben. Berechnet den **Arkustangens** von **y : x** entsprechend dem Vier-Quadranten-Polardiagramm. x und y können entweder beides Zahlen oder beides Kurven sein und das Resultat wird wieder eine Zahl oder eine Kurve sein. Ein Zahlen-Resultat wird in Grad

	(-180° bis +180°), ein Kurven-Resultat in Radiant (-Pi bis Pi) angegeben.
	
	Example:
	`a1 = 1`
	`s1 = ATan(a1) ; 45°`
	
	`a2 = 0`
	`s2 = ATan(a2) ; 0°`
	
	`; convert rad <-> deg`
	`; deg = 360 / (2*Pi) * rad`
	`; rad = (2*Pi) /360  * deg`
	
	`; alternatively to  function angle()`
	`x = -3+5j ; complex number`
	`deg = atan(imag(x), real(x)) ; deg = 120.96°`

Cos(a)	Berechnet den **Cosinus** von **a**. Der Resultattyp entspricht wieder **a**. Die Werte werden als **Grad interpretiert** (Vollkreis = 360°).
	
	Example:
	`a1 = 0 ; deg`
	`s1 = Cos(a1) ; 1`
	
	`a2 = 60 ; deg`
	`s2 = Cos(a2) ; 0.5`
	
	`; convert rad <-> deg`
	`; deg = 360 / (2*Pi) * rad`
	`; rad = (2*Pi) /360  * deg`

Exp(a)	Berechnet die **Exponentialfunktion** von **a**. Der Resultattyp entspricht wieder **a**.
	
	Example:
	`; Euler's number`
	`e = exp(1) ; 2.7183`
	`val = ln(e) ; 1`
	
	`;Discharge function of a capacitor`
	`R = 1E3  ; 1 kilo Ohm`
	`C = 100E-6  ; 100 micro Farad`
	`U = 10  ; Start voltage 10 V`
	`tsamp = 0.01 ; 100Hz sample interval`
	
	`; create a list with calucated voltages`
	`vlist = List()`
	`for t = 0 to 1 step tsamp`
	`  Uc = U * exp(-t/(R*C))`
	`  vlist()=Uc`
	`next`
	
	`; convert the list into a trace`
	`tr[V] = ConvToTrace(vlist, 1/tsamp)`
	
	`; calculate time for a defined voltage`
	`U1 = 4`
	`t1 = ln((U1/U))* (-R*C)`
	
	`; place cursort on position of t1`
	`SetCrs("Waveform 1","A", t1)`

Ln(a)	Berechnet den **natürlichen Logarithmus** von **a**. Der Resultattyp entspricht wieder **a**.

	Example:
	`; Euler's number`
	`e = exp(1) ; 2.7183`
	`val = ln(e) ; 1`
	
	`;Discharge function of a capacitor`
	`R = 1E3  ; 1 kilo Ohm`
	`C = 100E-6  ; 100 micro Farad`
	`U = 10  ; Start voltage 10 V`
	`tsamp = 0.01 ; 100Hz sample interval`
	
	`; create a list with calucated voltages`
	`vlist = List()`
	`for t = 0 to 1 step tsamp`
	`  Uc = U * exp(-t/(R*C))`
	`  vlist()=Uc`
	`next`
	
	`; convert the list into a trace`
	`tr[V] = ConvToTrace(vlist, 1/tsamp)`
	
	`; calculate time for a defined voltage`
	`U1 = 4`
	`t1 = ln((U1/U))* (-R*C)`
	
	`; place cursort on position of t1`
	`SetCrs("Waveform 1","A", t1)`

Log(a [, base])	Berechnet den **Logarithmus** (Basis 10). Verwenden sie den Parameter **base** um einen Logarithmus mit spezifischer Basis zu berechnen. Der Resultattyp entspricht wieder **a**.
	
	Example:
	`; base = 10 without additional parameter`
	`l1 =  log(100)  ; l1 = 2`
	`v1 = 10^l1  ; v1 = 100`
	
	`; base = 2`
	`l2 = log(64, 2) ; l2 = 6`
	`v2 = 2^l2  ; v2 = 64`
	
	`; amplifier calculation of gain in dB`
	`Uin = 1`
	`Uout = 20`
	`gain = 20*log(Uout/Uin) ; 26.021dB`

Sin(a)	Berechnet den **Sinus** von **a**. Der Resultattyp entspricht wieder **a**. Die Werte von **a** werden als **Grad interpretiert** (Vollkreis = 360°).
	
	Example:
	`a1 = 90 ; deg`
	`s1 = Sin(a1) ; 1`
	
	`a2 = 30 ; deg`
	`s2 = Sin(a2) ; 0.5`
	
	`; convert rad <-> deg`
	`; deg = 360 / (2*Pi) * rad`
	`; rad = (2*Pi) /360  * deg`

Tan(a)	Berechnet den **Tangens** von **a**. Der Resultattyp entspricht wieder **a**. Die Werte von **a** werden als **Grad interpretiert** (Vollkreis = 360°). **Example:** `a1 = 45 ; deg`

	`s1 = Tan(a1) ; 1`
	
	`a2 = 0 ; deg`
	`s2 = Tan(a2) ; 0`
	
	`; convert rad <-> deg`
	`; deg = 360 / (2*Pi) * rad`
	`; rad = (2*Pi) /360  * deg`

34.7 File Functions

CloseFile(a)	Schliesst die Datei (Handle) mit dem Namen **a**. Beispiel: Verwenden sie **CloseFile** mit **WriteLine** um eine neue Dateinummer zu starten falls **a** einen Hashtag '#' verwendet.
	
	Geöffnete Dateien werden automatisch am Ende der Formel geschlossen.
	
	Example:
	`filename = "meanMaxMean#.txt"`
	
	`for blockIdx = 0 to NBlks(c0A1) - 1 step 1.0`
	`  minVal = Min(c0A1.blockIdx)`
	`  maxVal = Max(c0A1.blockIdx)`
	`  meanVal = Mean(c0A1.blockIdx)`
	`  WriteLine(filename, Tabulator, "Min:", "Max:", "Mean:")`
	`  WriteLine(filename, Tabulator, minVal, maxVal, meanVal)`
	
	`  ; We close here the file to start a new file`
	`  CloseFile(filename)`
	`next`

Col	Schlüsselwort, welches für die Funktion **"StoreReadouts"** verwendet wird. Mit **Col** wird die entsprechende Spalte der Skalar Tabelle abgespeichert. Siehe auch das Schlüsselwort **Line**. **Example:** `StoreReadouts("readouts.txt", "Skalar_A 1", Col, 1)`
Comma	Schlüsselwörter für den Parameter **seperator** der Funktionen **WriteLine** und **WriteColumn**. Sie bestimmen das Trennzeichen der einzelnen Einträge.
CopyFile(srcFilepath, destFilepath, overwrite [, cancellable])	Kopiert die Datei mit dem Namen **srcFilepath** nach **destFilepath**. Mit dem Parameter **overwrite** vom Typ Boolean wir d angegeben, ob eine bereits existierende Datei überschrieben werden soll (**True**). Der optionale Parameter **cancellable** vom Typ Boolean gibt an ob der Kopierprozess abgebrochen werden kann oder nicht. Wenn die Datei erfolgreich kopiert wurde, gibt diese Funktion 1 zurück, ansonsten 0.
CreateDirectory(path)	Erstellt ein neues Verzeichnis mit dem Namen **path**. Diese Funktion generiert keine Rückgabewert.
DeleteDirectory(path)	Löscht ein existierendes Verzeichnis **path** mit allen Unterverzeichnissen und Dateien.

	Als Rückgabewert erhält man eine 1, wenn das Verzeichnis **path** erfolgreich gelöscht wurde, eine 0 falls das Verzeichnis nicht gefunden wurde.
	
	Alle Dateien und Unterverzeichnisse werden ohne Rückmeldung gelöscht!
	
	Example:
	`; Delete data folder in Experiment path`
	`variable = DeleteDirectory("data")`

DeleteFile(a)	Löscht eine oder mehrere Dateien welche in **a** definiert sind. Die Funktion bringt **False** (=0) zurück, wenn mind. eine Datei nicht gelöscht werden konnte (die Datei ist vielleicht im Signalquellen-Browser eingetragen und/oder Kurven davon sind in einer Kurvenanzeige vorhanden). **True** (=gültig) wird auch zurückgegeben, wenn Dateien nicht vorhanden sind.
	
	Example:
	`; Delete single file in data folder of Experiment`
	`DeleteFile("123.txt")`
	
	`; Delete single file in a specific location`
	`test = DeleteFile("C:\xxx\xxx\TranAX_4.1\xxx.exp\123.txt")`
	
	`; Delete multiple files from a list`
	`mylist = List()`
	`mylist() = "123.txt"`
	`mylist() = "456.txt"`
	`mylist() = "789.txt"`
	`test = DeleteFile(mylist)`

DirectoryExists(path)	Überprüft ob das Verzeichnis **path** existiert. Gibt **True** zurück wenn es existiert, ansonsten **False**.
	
	Example:
	`path = GetPath("data")`
	
	`if DirectoryExists(path) = true then`
	`  ; do something`
	`  pause("Exists!")`
	`endif`

File(filepath, index)	Die Funktion **File** gibt eine Kurve aus einer Datei zurück (Normalerweise *.tpc5). Es können die analogen Signale als auch die digitalen (Marker) Signale ausgelesen werden. Bei Multiblock-Aufnahmen kann auf die einzelnen Blöcke zugegriffen werden. Es wird immer nur ein Block ausgelesen. **filepath** ist ein String mit dem Namen der zu öffnenden Datei. Es werden nur Dateien mit der Erweiterung *.tpc5 oder *.tdp akzeptiert. Wenn kein Pfad mit dem Dateinamen angegeben wird, sucht das Programm die Datei im Verzeichnis **DATA** des **aktuellen Experiments**. Für eine Datei in einem anderen Verzeichnis innerhalb des aktuellen Experiments muss nicht die gesamte Pfadbezeichnung (z.B. "C:\User\USER-NAME\Documents\TransAS\EXPERIMENT.exp

	\Ref\NAME.tpc5") angegeben werden. Es genügt die Bezeichnung "..\Ref \NAME.tpc5".
	
	Index ist die Kurven-Nummer in der Datei (nicht zu verwechseln mit der Kanal-Nummer). 0 entspricht der ersten Kurve. **Beispiel:** File("dateiname.tpc5", 1) In diesem Beispiel wird also die zweite Signalkurve der Datei für die Berechnung verwendet.
	
	Mit **.Block** kann bei Multiblock-Aufnahmen auf die einzelnen Blöcke zugegriffen werden. Die Block-Nummer ist fakultativ. Wenn sie fehlt, wird immer Block 0 angenommen. Es wird immer nur ein Block ausgelesen. **Beispiel:** File("Dateiname", Index).Block
	
	Mit **'Marker** kann anstelle der Analog-Kurve (entsprechend dem Parameter **Index**) eine der beiden digitalen Spuren (Marker) ausgelesen werden. Dieser Parameter kann die Werte 1 oder 2 an-nehmen. Das Resultat besteht aus einer Kurve mit den Amplitudenwerten 0 oder 1. Marker sind nur vorhanden, wenn bei der ursprünglichen Aufnahme im Control-Panel "Averaging" nicht auf 16Bit gestellt war. **Beispiel:** File("crash.tpc5",0)'1
	
	Example:
	`filepath = "signal.tpc5"`
	`variable = File(filepath, 0)`

FileExist(a)	Überprüft ob die Datei **a** existiert. Gibt **True** zurück wenn es existiert, ansonsten **False**.
	
	Example:
	`if FileExist("xy.tpc5") = True then`
	`  ;do something`
	`endif`

FileIndexExists(a, index)	Überprüft ob ein **index** (0…) in **file** existiert.
	Bei **file** kann als String der Name einer *.tpc5 Datei oder auch eine **Referenz** übergeben werden (Bsp. Ref1).
	Der Rückgabewert ist ein Skalar vom Typ Integer, 1 bedeutet der Kanal oder die Messkurve **index** ist vorhanden, 0 bedeutet nicht vorhanden.
	
	Siehe auch die Funktion **NTracesInFile**. Für Referenzen kann auch die Funktion **Length** verwendet werden.
	Beide Funktionen geben die Anzahl der Kurven zurück.
	
	Example:
	`fname$ = "test.tpc5"; Signale generieren`
	`tr1 = Noise(1, 1E3, 1E3)`
	`tr2 = Noise(2, 1E3, 1E3)`
	`Save(fname$, tr1, tr2) ; Kurven speichern`
	
	`for a = 0 to 10 step 1 ; hier bis zu 11 Kurven`
	`  val = FileIndexExists (fname$, a)`
	`  if val = True then  ; ist Kurve vorhanden ?`
	`  tr = File(fname$, a) ; Kurve kann geholt werden`
	`  ; weitere Berechnungen`

	`  endif` `next`

FileNameInc(filepath [, nrOfDigits])	Inkrementiert den Zähler eines Dateinamens **filepath** welcher einen Hashtag '#' enthält. Der optionale Parameter **nrOfDigits** definiert wieviele Anzahl Stellen im neuen Dateinamen vorhanden sein sollen.
	
	Example:
	`; Sine wave signal`
	`tr = Sinus(10,50,0,1E3,1E3)`
	
	`; File name template`
	`fnameTemplate = "myTrace_#.tpc5"`
	
	`; create 11 files`
	`for a = 0 to 10 step 1.0`
	`  fname = FileNameInc(fnameTemplate)`
	`  Save(fname, tr)`
	`  tr = tr*1.01`
	`next`

GetDirectoryName(filepath)	Gibt den Verzeichnisnamen von **filepath** zurück. **Example:** `dirName = GetDirectoryName("C:\Test\data.tpc5") ; dirName = "C:\Test"`
GetFileExtension(filepath [, knownExtensions])	Gibt die Dateierweiterung der Datei **filepath** zurück. Der optionale Parameter **knownExtensions** kann dazu verwendet werden, um spezielle Dateitypen mit zwei oder mehr Punkten zu erkennen. Ein Beispiel dazu ist die *.tps.xml Hardwareeinstellungen. **Example:** `extension = GetFileExtension("C:\Test\data.tpc5") ; extension = ".tpc5"`

GetFileInfo(filepath)	Gibt alle relevanten Dateiparameter von **filepath** zurück als Dictionary. Die dazugehörigen Schlüssel sind wie folgt:
	
	Zeitinformationen
	"creationTime", "creationTimeUtc", "lastAccessTime", "lastAccessTimeUtc", "lastWriteTime", "lastWriteTimeUtc"
	Attribute
	"isReadOnly", "isHidden", "isDirectory", "isOffline", "isCompressed", "isEncrypted", "isNormal"
	Grösse, Dateiname und Pfad
	"sizeInBytes", "extension", "name", "directoryName", "fullName"
	
	Example:
	`; Create a file with one trace`
	`fname = "mytrace.tpc5"`
	`tr = Sinus(10,50,0, 1E3, 1E3)`
	`Save(fname, tr)`
	
	`; Read file info`
	`info = GetFileInfo(fname)`
	
	`key = "sizeInBytes"`
	`if HasKey(info, key) = 1 then`

	`  size = info(key)`
	`else`
	`  size = notdefined`
	`endif`

GetFilename(filepath, [, withoutExtension])	Gibt den Dateinamen der Datei **filepath**, ohne Pfad. Der optionale Parameter **withoutExtension** gibt an, ob der Dateiname die Endung beinhalten soll.
	
	Example:
	`fname1 = GetFilename("C:\Test\data.tpc5", false) ; fname1 = "data.tpc5"`
	`fname2 = GetFilename("C:\Test\data.tpc5", true) ; fname2 = "data"`

GetFiles(srcPath [, pattern])	Gibt ein Array mit allen vorhandenen Dateien in **srcPath** zurück. Der optionale Parameter **pattern** definiert ob **GetFiles** nur gewisse Dateiendungen berücksichtigen soll (Beispiel: "*.tpc5").
	
	Die Funktion gibt die Dateien von Unterverzeichnissen nicht zurück.
	
	Example:
	`; analyze and add all *.tpc5 files from a directory`
	`fileArray = GetFiles(GetPath("Data"), "*.tpc5")`
	`nItems = Length(fileArray) ; number of files found`
	`SetMessage("Found "+ nItems +" items(s)", True)`
	
	`sizeByte = 0`
	`for each item in fileArray`
	`  ;do some analysis`
	`  info = GetFileInfo(item)`
	
	`  key = "sizeInBytes"`
	`  if HasKey(info, key) = 1 then`
	`  val = info(key)`
	`  sizeByte += val`
	`  endif`
	`next`
	
	`Pause("File info:", nItems, sizeByte)`

GetHarmonics(a, nrOfHarmonics)	Berechnet die Harmonischen von **a**. Der Parameter **nrOfHarmonics** definiert die Anzahl der Harmonischen die Berechnet werden sollen. Rückgabewert **variable** ist vom Typ Dictionary. Die Schlüssel sind wie folgt:
	
	percent, peak, rms, decibel, frequency, phase
	
	Example:
	`tr = Sinus(10,50,0,10E3,10E3)  ; create a noisy 50Hz signal`
	`tr = tr + Noise(2,tr)`
	
	`dictHarm = GetHarmonics(tr, 6)`
	
	`arr = dictHarm("peak")`
	`peak1st = arr(0)`
	`arr = dictHarm("frequency")`
	`Freq1st = arr(0)`

GetNewestFile(srcPath [, pattern])	Gibt die neuste Datei innerhalb des Verzeichnisses **srcPath** zurück. Der optionale Parameter **pattern** kann die zurückgegebenen Dateien anhand von Dateiendungen einschränken.
	
	Example:
	`sourcePath = GetPath("data")`
	`fnewest = GetNewestFile(sourcePath, "*.tpc5")`
	
	`tr = File(fnewest,0)`
	`; do something with this trace`

GetPath(a)	Gibt Experiment spezifische Verzeichnisse zurück. Mögliche Werte für den Parameter **a** sind: **experiment**, **experimentset**, **data**, **expr**, **commonexperimentpath**, **settingspath**. Des weiteren werden die **Standard Windows Umgebungsvariablen** wie z.B: %appdata% oder %programdata% ebenfalls unterstützt. **Example:** `dataPath = GetPath("data")`
Line	Schlüsselwort, welches für die Funktion **"StoreReadouts"** verwendet wird. Mit **Line** wird die entsprechende Zeile der Skalar Tabelle abgespeichert. Siehe auch das Schlüsselwort **Col**. **Example:** `StoreReadouts("readouts.txt", "Skalar_A 1", Line, 1)`
NTracesInFile(filepath)	Gibt eine Zahl zurück, welche die Anzahl der enthaltenen Kurven in **filepath** angibt. Falls die Datei **filepath** nicht vorhanden ist, meldet der Formel Editor einen Fehler. Es werden nur Dateien mit der Erweiterung *.tpc5 oder *.tdp akzeptiert. Bezüglich Dateipfad gilt das gleiche, wie bei der Funktion **File**. **Example:**

```
; simple example
if FileExist("myfile.tpc5") = True  then
    Ntrace1=NTracesInFile("xy.tpc5")
else
    Ntrace=0
endif


; advanced example, read all blocks from all traces in a file
fname = "myfile.tpc5"
; valid filename and path?
if FileExist(fname) = false then
    Pause("File not found")
    EndFormula
endif


minVal = List() ; create a list

nTr = NTracesInFile(fname)    ; number of traces in file
```

	`for a = 0 to nTr-1 step 1.0  ; go through each trace`
	`  minVal = List()`
	`  for each item in  enBlocks(file(fname, 0),false)  ; go through each block`
	`  ; do something with each block`
	`  minVal() = Min(item) ; write min of each block into a list`
	`  next`
	`next`

PathCombine(path*)	Kombiniert mehrere Zeichenfolgen **path** zu einer gültigen Pfadzeichenfolge (Fügt Backslashes zwischen den Elementen hinzu).
	
	Example:
	`Mainpath = "C:\Users"`
	`User  = "Default"`
	`Directory = "Desktop"`
	
	`NewPath = PathCombine(Mainpath, User, Directory) ;NewPath = C:\Users\Default\Desktop`

ReadLine(filepath, separator)	Liest eine komplette Zeile aus der Textdatei **filepath**. Die Werte von der gelesenen Linie werden anhand des Trennzeichens **separator** aufgetrennt. Rückgabewert ist ein Array mit allen Werten.
	
	Example:
	`; create an example file`
	`filepath = "text.txt"`
	`WriteLine(filepath, Tabulator, "Voltage1", 10)`
	`WriteLine(filepath, Tabulator, "Voltage2", 10.5)`
	`CloseFile(filepath)`
	`lineArray1 = ReadLine(filepath, Tabulator) ; lineArray1 = [Voltage1, 10]`
	`lineArray2 = ReadLine(filepath, Tabulator) ; lineArray2 = [Voltage2, 10.5]`
	` CloseFile(filepath)`

RenameFile(srcFilepath, destFilepath, overwrite)	Die existierende Datei **srcFilepath** wird umbenennt zu **destFilepath**. Mit dem Parameter **overwrite** auf **True** wird eine existierende Datei überschrieben.
Save(filepath, [options,] trace*) **Save(filepath, spectrum*)** **Save(filepath, filename*)** **Save(filepath, array)** **Save(filepath, [options,] list)** **Save(filepath, [options,] list, nameList|nameArray)**	**filepath** ist ein String mit dem Namen der zu erzeugenden Datei. Der Dateiname muss die Erweiterung *.tpc5 haben, wenn die gespeicherten Kurven wieder mit TranAX analysiert werden sollen. Wenn ein **"#" (Hash/Pound Symbol)** am Ende des Dateinamens ist, wird dieses Zeichen durch eine **fortlaufende Nummer** ersetzt. (z.B. "Check#.tpc5" ergibt "Check001.tpc5", "Check002.tpc5", ...). Bezüglich Dateipfad gilt das gleiche, wie unter "File(....)". **trace** bezeichnet die Kurven-Variable die gespeichert werden soll. Es können auch mehrere durch Kommas getrennte Kurven angegeben werden. Mit dem optionalen Parameter **options** können zusätzliche Parameter definiert werden, z. B. das Trennzeichen für den ASCII-Export. **Ascii export** "Conversion": "Marker", "Both"

	"Separator": "", ";", "," "DecimalPoint": ". (Point)", ", (Comma)" **Tpc export (TransAS 2)** "FilePerChannel": "true", "false" **SEG-Y export (geophysical data)** "Endianness": "LittleEndian", "BigEndian" Wenn die Datei **erfolgreich gespeichert** werden konnte, gibt die **Funktion 1 (True)**, im Fehlerfall 0 (False) zurück. Eine bereits existierende Datei mit gleichem Namen wird ohne Warnung überschrieben. **filepath** ist ein String mit dem Namen der zu erzeugenden Datei. Der Dateiname muss die Erweiterung *.tpc5s haben, wenn die gespeicherten Kurven wieder mit TranAX FFT analysiert werden sollen. Wenn ein **"#" (Hash/Pound Symbol)** am Ende des Dateinamens ist, wird dieses Zeichen durch eine **fortlaufende Nummer** ersetzt. (z.B. "Check#.tpc5s" ergibt "Check001.tpc5s", "Check002.tpc5s", ...). Bezüglich Dateipfad gilt das gleiche, wie unter "File(....)". **spectrum** bezeichnet die **FFT Kurven-Variable** die gespeichert werden soll. Es können auch mehrere durch Komma getrennte Kurven angegeben werden. Wenn die Datei **erfolgreich gespeichert** werden konnte, gibt die **Funktion 1 (True)**, im Fehlerfall 0 (False) zurück. Eine bereits existierende Datei mit gleichem Namen wird ohne Warnung überschrieben. **filepath** ist ein String mit dem Namen der zu erzeugenden Datei. Der Dateiname muss die Erweiterung *.tpc5 haben, wenn die gespeicherten Kurven wieder mit TranAX analysiert werden sollen. Wenn ein **"#" (Hash/Pound Symbol)** am Ende des Dateinamens ist, wird dieses Zeichen durch eine **fortlaufende Nummer** ersetzt. (z.B. "Check#.tpc5" ergibt "Check001.tpc5", "Check002.tpc5", ...). Bezüglich Dateipfad gilt das gleiche, wie unter "File(....)". **filename** bezeichnet die Dateinamen als String, welche in die neue Kurvendatei gespeichert werden sollen. Es können auch mehrere durch Kommas getrennt mehrere Dateien angegeben werden.

	Wenn die Datei **erfolgreich gespeichert** werden konnte, gibt die **Funktion 1 (True)**, im Fehlerfall 0 (False) zurück. Eine bereits existierende Datei mit gleichem Namen wird ohne Warnung überschrieben. **filepath** ist ein String mit dem Namen der zu erzeugenden Datei. Der Dateiname muss die Erweiterung *.tpc5 haben, wenn die gespeicherten Kurven wieder mit TranAX analysiert werden sollen. Wenn ein **"#" (Hash/Pound Symbol)** am Ende des Dateinamens ist, wird dieses Zeichen durch eine **fortlaufende Nummer** ersetzt. (z.B. "Check#.tpc5" ergibt "Check001.tpc5", "Check002.tpc5", ...). Bezüglich Dateipfad gilt das gleiche, wie unter "File(....)". **array** bezeichnet die Kurven-Variablen die gespeichert werden sollen. Wenn die Datei **erfolgreich gespeichert** werden konnte, gibt die **Funktion 1 (True)**, im Fehlerfall 0 (False) zurück. Eine bereits existierende Datei mit gleichem Namen wird ohne Warnung überschrieben. **filepath** ist ein String mit dem Namen der zu erzeugenden Datei. Der Dateiname muss die Erweiterung *.tpc5 haben, wenn die gespeicherten Kurven wieder mit TranAX analysiert werden sollen. Wenn ein **"#" (Hash/Pound Symbol)** am Ende des Dateinamens ist, wird dieses Zeichen durch eine **fortlaufende Nummer** ersetzt. (z.B. "Check#.tpc5" ergibt "Check001.tpc5", "Check002.tpc5", ...). Bezüglich Dateipfad gilt das gleiche, wie unter "File(....)". **list** bezeichnet die Kurven-Variablen die gespeichert werden sollen. Mit dem optionalen Parameter **options** können zusätzliche Parameter definiert werden, z. B. das Trennzeichen für den ASCII-Export. **Ascii export** "Conversion": "Marker", "Both" "Separator": "", ";", "," "DecimalPoint": ". (Point)", ", (Comma)" **Tpc export (TransAS 2)** "FilePerChannel": "true", "false"

SEG-Y export (geophysical data)
"Endianness": "LittleEndian", "BigEndian"

Wenn die Datei **erfolgreich gespeichert** werden konnte, gibt die **Funktion 1 (True)**, im Fehlerfall 0 (False) zurück.

> Eine bereits existierende Datei mit gleichem Namen wird ohne Warnung überschrieben.

filepath ist ein String mit dem Namen der zu erzeugenden Datei. Der Dateiname muss die Erweiterung *.tpc5 haben, wenn die gespeicherten Kurven wieder mit TranAX analysiert werden sollen.
Wenn ein **"#" (Hash/Pound Symbol)** am Ende des Dateinamens ist, wird dieses Zeichen durch eine **fortlaufende Nummer** ersetzt. (z.B. "Check#.tpc5" ergibt "Check001.tpc5", "Check002.tpc5", ...). Bezüglich Dateipfad gilt das gleiche, wie unter "File(....)".

nameList vom Typ list oder **nameArray** vom Typ array bezeichnen die **Kurvennamen** die gespeichert werden sollen.\

Mit dem optionalen Parameter **options** können zusätzliche Parameter definiert werden, z. B. das Trennzeichen für den ASCII-Export.
Ascii export
"Conversion": "Marker", "Both"
"Separator": "", ";", ","
"DecimalPoint": ". (Point)", ", (Comma)"
Tpc export (TransAS 2)
"FilePerChannel": "true", "false"
SEG-Y export (geophysical data)
"Endianness": "LittleEndian", "BigEndian"

Wenn die Datei **erfolgreich gespeichert** werden konnte, gibt die **Funktion 1 (True)**, im Fehlerfall 0 (False) zurück.

> Eine bereits existierende Datei mit gleichem Namen wird ohne Warnung überschrieben.

Example:

```
; 1. simple example
v[m/s] = Smooth(c0A1,200)
Save_OK = Save("Velocity.tpc5", v)
; File may be open
if Save_OK = False then
    ; Use a different name
    Save("Velocity-2.tpc5", v)
endif

; 2. simple example
; the # will be replaced with a number for each calculation
```

```
save("myTrace#.tpc5", c0A1) ; myTrace001.tpc5, etc.

; 3. simple example
; saves just 1. block
; TPC5 file
Save("save1_Block1.tpc5", c0A1, c0A2, c0A3)
; ASCII file
Save("save1_Block1.asd", c0A1, c0A2, c0A3)

; 4. simple example
; save all blocks
; TPC5 file
Save("save1_AllBlocks.tpc5", "0A1", "0A2", "0A3")
; ASCII file
Save("save1_AllBlocks.asd", "0A1", "0A2", "0A3")


; ### Advanced exmaples ###
; get 1. block from hardware channels
trace1 = c0A1.0
trace2 = c0A2.0
trace3 = c0A3.0
; define separator for ASCII export
options = Array("Separator", ",")

; export ASCII with customiced sperator
Save("save2.asd", options, trace1, trace2, trace3)

; save some specific parts from hardware channels
; the values withing [] defines block number
; and data reduction like Skip, Average or MinMax
Save("save3a.tpc5", "0A1-2, 0A4", "0A3[1-5]", "0B1[0-5:Skip5]")
Save("save3b.tpc5", "0A1-2, 0A4", "0A3[1-5]", "0B1[0-
5:Average10]")
Save("save3c.tpc5", "0A1-2, 0A4", "0A3[1-5]", "0B1[0-5:Min-
Max12]")


; save multiple filies into one file
Save("save4.tpc5", "save3a.tpc5", "save3b.tpc5", "save3c.tpc5")


; use lists and strings mixed examples
listTraces = List(trace1, trace2, trace3)
Save("save5.tpc5", listTraces)
Save("save5.asd", options, listTraces)

listNames = List(GetChName(trace1), GetChName(trace2),
GetChName(trace3))
val1 = Save("save6.tpc5", listTraces, listNames)
val2 = Save("save6.asd", options, listTraces, listNames)

arrChanDesc = Array("0A1-2", "0B1[0-5:Skip5]")
Save("save7.asd", arrChanDesc)
listNameMisc = List(GetChName(c0B2), GetChName(c0B3),
GetChName(c0B1), GetChName(trace1), GetChName(trace2),
GetChName(trace3))
listMisc = List("0B2", "0B3", "0B1[0-5:Skip5]", trace1, trace2,
trace3)
Save("save8.tpc5", listMisc, listNameMisc)


; exmaples for FFT traces
traceFFT1 = FFT(trace1)
traceSpec1 = ConvToSpectrumTrace(traceFFT1)
traceFFT2 = FFT(trace2)
traceSpec2 = ConvToSpectrumTrace(traceFFT2)
traceFFT3 = FFT(trace3)
traceSpec3 = ConvToSpectrumTrace(traceFFT3)
Save("saveFFT1.tps5", traceSpec1, traceSpec2)
```

	; Channels have to be opened in a FFT window. ; Make sure that you only save one block with the time window otherwise it will not work. Save("saveFFT2.tps5", "0A1-3")
Semicolon	Schlüsselwörter für den Parameter **seperator** der Funktionen **WriteLine** und **WriteColumn**. Sie bestimmen das Trennzeichen der einzelnen Einträge.
Space	Schlüsselwörter für den Parameter **seperator** der Funktionen **WriteLine** und **WriteColumn**. Sie bestimmen das Trennzeichen der einzelnen Einträge.
StoreReadouts(filepath, name, type, number [, results*])	Speichert die durch Formeln ermittelten Einzelwert-Resultate **results** in eine Text-Datei mit dem Namen Filename. Speichert die Resultate aus der Skalar-Tabelle **name** in eine Text-Datei mit dem Namen **filepath**. Mit den Parametern **type** (Line/Col) und der entsprechenden Nummer **number** (**LinNr** oder **ColNr**) können die Zellen in der Skalar-Tabelle bestimmt werden. Optional können auch noch die durch Formeln ermittelten Einzelwert-Resultate **results** angehängt werden. Bezüglich Dateipfad gilt das gleiche, wie unter **File** Funktion beschrieben. Wenn ein "#" am Ende des Dateinamens ist, wird dieses Zeichen durch eine fortlaufende Nummer ersetzt. **Example:** StoreReadouts(Filename, Results*) StoreReadouts(Filename, ScalarTableName, Line | Column, LineNr | ColNr[, Results]*)
Tabulator	Schlüsselwörter für den Parameter **seperator** der Funktionen **WriteLine** und **WriteColumn**. Sie bestimmen das Trennzeichen der einzelnen Einträge.
Write(filepath, value*)	Schreibt die Parameter **value** in die Datei **filepath**. Es werden jeweils neue Daten und Zeilen an die existierende Datei angefügt (Append Modus). Das Schreiben der Zeichenfolge "\r"C und "\n"C generiert einen Zeilenumbruch.
WriteColumn(filepath, separator, value*)	Diese Funktion macht im Wesentlichen das gleiche wie **WriteLine**. Die einzelnen Elemente eines Arrays und/oder einer Liste werden dabei in derselben Kolonne abgelegt. Für jeden einzelnen Parameter **value** entsteht also eine Kolonne. Beim wiederholten Aufrufen von **WriteColumn** werden die Kolonnen unten mit den neuen Werten erweitert. Da dies meist auch zu Unübersichtlicheit führt, sollte der Aufruf nur einmal erfolgen. Auch ist diese Funktion kaum geeignet, wenn nur Einzel-Werte vorkommen. Dies ergäbe den gleiche Datei-Inhalt wie bei WriteLine. **Example:** WriteColumn(fname ,separator, value*)
WriteLine(filepath, separator, value*)	Schreibt den/die Wert(e) **value** in die Text-Datei **filepath** (normalerweise im Data-Verzeichnis des aktuellen Experiments). Ein "**#**" hinter dem Datei-Namen wird durch eine fortlaufende

	Nummer ersetzt. Dadurch kann ein Überschreiben von bestehenden Dateien verhindert werden. **separator** kann ein beliebiger String sein (auch nur ein Einzelzeichen z.B. Tabulator). Damit wird die Trennung zwischen den Werten festgelegt. **value*** sind Einzel-Werte oder Strings. Diese können auch vorab berechnet worden sein (Namen unter "Results"). **value** kann auch als Array oder List deklariert werden *). Falls die Text-Datei bereits existiert, werden beim ersten Aufruf dieser Funktion alle Einträge ohne Warnung gelöscht. Bei nachfolgenden **WriteLine** Aufrufen (beim aktuellen Abarbeiten der Formel-Kommandos) wird jeweils eine weitere Linie angehängt. Falls die Text-Datei noch nicht existiert, wird sie unter dem Namen **filepath** erzeugt. **Example:** `WriteLine(fname ,separator, value*)`

34.8 Filter Functions

BandPass(a, type, [, ripple] order, freq, bandwidth)	Ein Bandpass sperrt tiefere und höhere Frequenzbereiche, lässt den Mittelbereich durch. **a** ist eine Kurve welche gefiltert wird. **type** spezifiziert den **Filtertyp**:
	
	• Bessel
	• Butter (Butterworth)
	• Cheby (Tschebyscheff). Bei **Tschebyscheff** muss noch der Parameter **ripple** (Welligkeit im Durchlassbereich in dB) angegeben werden
	
	order spezifiziert die **Ordnung** (1 .. 9) des Filters. **freq** ist die **Mittenfrequenz** in Hz des Bandpasses oder der Bandsperre, **bandwidth** ist die Bandbreite in Hz.
	
	Der Rückgabewert **variabel** wieder eine Kurve.
	
	Example:
	`tr = c0A1`
	`trfilter = BandPass(tr, Bessel, 4, 10E3, 1E3)`

BandStop(a, type [, ripple], order, freq, bandwidth)	Eine Bandsperre lässt tiefere und höhere Frequenzbereiche durch, sperrt den Mittelbereich. **a** ist eine Kurve welche gefiltert wird. **type** spezifiziert den **Filtertyp**: • Bessel • Butter (Butterworth) • Cheby (Tschebyscheff). Bei **Tschebyscheff** muss noch der Parameter **ripple** (Welligkeit im Durchlassbereich in dB) angegeben werden **order** spezifiziert die **Ordnung** (1 .. 9) des Filters. **freq** ist die **Mittenfrequenz** in Hz des Bandpasses oder der Bandsperre, **bandwidth** ist die Bandbreite in Hz.

	Der Rückgabewert **variabel** wieder eine Kurve.
	
	Example:
	`tr = c0A1`
	`trfilter = BandStop(tr, Bessel, 4, 10E3, 1E3)`

Bessel	Schlüsselwort **type** zu den Funktionen **LowPass**, **HighPass**, **BandPass** und **BandStop**.
Butter	Schlüsselwort **type** zu den Funktionen **LowPass**, **HighPass**, **BandPass** und **BandStop**.
Cheby	Schlüsselwort **type** zu den Funktionen **LowPass**, **HighPass**, **BandPass** und **BandStop**.

HighPass(a, type [, ripple], order, freq)	Ein Hochpass (HighPass) Filter schneidet tiefe Frequenzanteile ab. Der Parameter **a** ist eine Kurve welche gefiltert wird. **type** spezifiziert den **Filtertyp**:
	
	• Bessel,
	• Butter (Butterworth)
	• Cheby (Tschebyscheff). Bei **Tschebyscheff** muss noch der Parameter **ripple** (Welligkeit im Durchlassbereich in dB) angegeben werden
	
	Der Parameter **order** spezifiziert die **Ordnung** (1 .. 9) des Filters. **freq** ist die **Grenzfrequenz** in Hz Der Rückgabewert **variable** ist wieder eine Kurve.
	
	Example:
	`tr = Sinus(10,50,0,1E3,1E3)`
	`tr = tr + Noise(2,tr)`
	
	`trLO = LowPass(tr, Bessel, 4, 80)`
	`trHP = HighPass(tr, Bessel, 4, 80)`
	
	`; optional to prevent phase shift`
	`trLoNoShift = Flip(trLO)`
	`trLoNoShift = LowPass(trLoNoShift, Bessel, 4, 80)`
	`trLoNoShift = Flip(trLoNoShift)`

LowPass(a, type [, ripple], order, freq)	Ein Tiefpass (LowPass) Filter schneidet hohe Frequenzanteile ab. Der Parameter **a** ist eine Kurve welche gefiltert wird. **type** spezifiziert den **Filtertyp**:
	
	• Bessel,
	• Butter (Butterworth)
	• Cheby (Tschebyscheff). Bei **Tschebyscheff** muss noch der Parameter **ripple** (Welligkeit im Durchlassbereich in dB) angegeben werden
	
	Der Parameter **order** spezifiziert die **Ordnung** (1 .. 9) des Filters. **freq** ist die **Grenzfrequenz** in Hz Der Rückgabewert **variable** ist wieder eine Kurve.
	
	Example:
	`tr = Sinus(10,50,0,1E3,1E3)`
	`tr = tr + Noise(2,tr)`
	
	`trLO = LowPass(tr, Bessel, 4, 80)`

	`trHP = HighPass(tr, Bessel, 4, 80)`
	
	`; optional to prevent phase shift`
	`trLoNoShift = Flip(trLO)`
	`trLoNoShift = LowPass(trLoNoShift, Bessel, 4, 80)`
	`trLoNoShift = Flip(trLoNoShift)`

Median(a, width)	Die Funktion berechnet jeweils aus **width** Abtastwerten den **Median** von **a**. Das Resultat ist wieder eine Kurve. Mit dieser Funktion können einzelne (Stör-)Spitzen aus einer Kurve eliminiert werden.
	
	Example:
	`tr = c0A1`
	`trfilter = Median(tr, 50)`

Smooth(a, width)	Die Funktion berechnet den **gleitenden Mittelwert** von **a** über **width** Abtastwerte. Der Rückgabewert **variable** ist wieder eine Kurve.
	
	Example:
	`tr = c0A1`
	`trfilter = Smooth(tr, 50)`

34.9 Layout Waveform

BottomAxis	Schlüsselwort für den Parameter **orientation** in der Funktion **SetXRange** verwendet.
	
	Example:
	`; Sets the x-axis to +/- 10 seconds`
	`SetXRange("Waveform 1", BottomAxis, 0, -10, 10)`

GetWaveforms(name)	Liest die vorhanden Kurven aus einem Waveform.
	Der Parameter **name** ist der Name **einer vorhandenen Kurvenanzeige**. Rückgabewert **variable** ist dann eine **Liste mit Dictionaries**. Die vorhandenen Schlüssel sind **"name"** und **"filepath"** wenn es sich um eine Kurve aus einer Datei handelt.
	
	Example:
	`nameWaveForm = "Waveform 1"`
	`dictT = GetWaveforms(nameWaveForm)`
	
	`for each item in dictT`
	`  key = "filepath"`
	`  if HasKey(item, key) = 1 then`
	`  filepath = item(key)`
	`  endif`
	
	`  key = "name"`
	`  if HasKey(item, key) = 1 then`
	`  nameTrace = item(key)`
	`  endif`
	
	`  Pause("Traces in Waveform", nameWaveForm, filepath, name-`
	`Trace)`
	
	`next`

GetXRange(name, orientation, axisIdx)	Liest das X-Achsen-Intervall der Kurvenanzeige **name**. Rückgabewert **variable** ist ein Array mit zwei Zahlen, welche die linke und die rechte Begrenzung darstellen. Die beiden Parameter

	orientation und **axisIdx** sind für zukünftige Erweiterungen reserviert und können auf 0 gesetzt werden.
	
	Example:
	`rangeX = GetXRange("Waveform 1", 1, 0)`
	`rangeY = GetYRange("Waveform 1", 0, LeftAxis, 0) ; 1st Area`
	
	`xMin = rangeX(0)`
	`yMax = rangeY(1)`

GetYRange(name, areaIdx, orientation, axisIdx)	Liest das Y-Achsen-Intervall des Bereiches **areaIdx** der Kurvenanzeige **name**. Rückgabewert **variable** ist ein Array mit zwei Zahlen welche die linke und die rechte Begrenzung darstellen. Für den Parameter **orientation** verwendet man eines der beiden Schlüsselwörter **LeftAxis** oder **RightAxis**. **areaIdx** bezeichnet den entsprechenden Bereich im Kurvenanzeige-Fenster, beginnend mit 0 von oben. **axisIdx** ist in den meisten Fällen 0. Sind mehrere Y-Achsen vorhanden, wird dieser Index entsprechenden von innen nach aussen inkrementiert. Die Werte entsprechen den jeweiligen physikalischen Einheiten.
	
	Example:
	`rangeX = GetXRange("Waveform 1", 1, 0)`
	`rangeY = GetYRange("Waveform 1", 0, LeftAxis, 0) ; 1st Area`
	
	`xMin = rangeX(0)`
	`yMax = rangeY(1)`

LeftAxis	Schlüsselwörter für den Parameter **orientation** in der Funktion **SetYRange**.
	
	Example:
	`; Sets the y range of the first left axis to +/- 5`
	`SetYRange("Waveform 1", 0, LeftAxis, 0, -5, 5)`

RightAxis	Schlüsselwörter für den Parameter **orientation** in der Funktion **SetYRange**.
	
	Example:
	`; Sets the y range of the first right axis to +/- 5`
	`SetYRange("Waveform 1", 0, RightAxis, 0, -5, 5)`

SetXRange(name, orientation, axisIdx, min, max)	Setzt das X-Achsen-Intervall der Kurvenanzeige **name** auf die Werte **min** und **max**. **min** und **max** können auch vertauscht sein, d.h. der kleinere Wert entspricht immer der linken Begrenzung. Die beiden Parameter **orientation** und **axisIdx** sind für zukünftige Erweiterungen reserviert.
	
	Example:
	`; auto fit the waveform to a random trace`
	`amp = Random(10,20)`
	`samples = Random(1E3, 10E3)`
	
	`tr = Sinus(amp,50,0,1E3,samples)`
	`tr = tr + Noise(2,tr)`
	
	`t1 = TBegin(tr)`
	`t2 = TEnd(tr)`
	
	`maxVal = Max(tr)`
	`minVal = Min(tr)`

	`; change X-axis of waveform 1 form -10 to +5 seconds`
	`SetXRange("Waveform 1", BottomAxis, false, t1-0.1, t2+0.1)`
	`SetYRange("Waveform 1", LeftAxis, false, false, minVal-2, maxVal+2)`

SetYRange(name, areaIdx, orientation, axisIdx, min, max)	Setzt das Y-Achsen-Intervall des Bereiches **areaIdx** der Kurvenanzeige **name** auf die Werte **min** und **max**. Für den Parameter **orientation** verwendet man eines der beiden Schlüsselwörter- **LeftAxis** oder **RightAxis**. **areaIdx** bezeichnet den entsprechenden Bereich im Kurvenanzeige-Fenster, beginnend mit 0 von oben. **axisIdx** ist in den meisten Fällen 0. Sind mehrere Y-Achsen vorhanden, wird dieser Index entsprechenden von innen nach aussen inkrementiert. **min** und **max** können auch vertauscht sein, d. h. der kleinere Wert entspricht immer der unteren Begrenzung. Die Werte entsprechen den jeweiligen physikalischen Einheiten.
	
	Example:
	`; auto fit the waveform to a random trace`
	`amp = Random(10,20)`
	`samples = Random(1E3, 10E3)`
	
	`tr = Sinus(amp,50,0,1E3,samples)`
	`tr = tr + Noise(2,tr)`
	
	`t1 = TBegin(tr)`
	`t2 = TEnd(tr)`
	
	`maxVal = Max(tr)`
	`minVal = Min(tr)`
	
	`; change X-axis of waveform 1 from -10 to +5 seconds`
	`SetXRange("Waveform 1", BottomAxis, false, t1-0.1, t2+0.1)`
	`SetYRange("Waveform 1", LeftAxis, false, false, minVal-2, maxVal+2)`

34.10 Measurement Flow Control

mfcIsCalculating()	Prüft ob der Measurement Flow Control (MFC) aktiv.
	
	Rückgabewert **variable** ist vom Typ Integer: 1 = MFC ist aktiv, wird ausgeführt 0 = MFC ist inaktiv, wurde gestoppt
	
	Example:
	`; create an MFC with two tasks, both enbaled`
	`; with an delay of 5s as executable function`
	
	`; start Measurement Flow Control`
	`mfcstart()`
	
	`; do some code`
	`delay(2)`
	
	`; stop the second task`
	`val = mfcStopTask("Task 2")`
	`if val = 0 then`
	`  Pause("Task not found!")`
	`endif`
	
	`; check for aktive MFC`
	`status = mfcIsCalculating()`

	`if status = 1 then`
	`  ; stop Measurement Flow Control`
	`  retvalStop = mfcStop(false, true)`
	`endif`
	
	`; ckeck again status of ative MFC`
	`status = mfcIsCalculating()`
	`if status = 0 then`
	`  Pause("MFC stopped!")`
	`endif`

mfcStart()	Startet den Measurement Flow Control (MFC). Dabei wird zuerst die Initialize Tasks und anschliessen alle anderen Tasks gemäss Konfiguration gestartet. Diese Funktion hat keinen zusätzlichen Parameter oder Rückgabewert.
	
	Example:
	`; create an MFC with two tasks, both enbaled`
	`; with an delay of 5s as executable function`
	
	`; start Measurement Flow Control`
	`mfcstart()`
	
	`; do some code`
	`delay(2)`
	
	`; stop the second task`
	`val = mfcStopTask("Task 2")`
	`if val = 0 then`
	`  Pause("Task not found!")`
	`endif`
	
	`; check for aktive MFC`
	`status = mfcIsCalculating()`
	
	`if status = 1 then`
	`  ; stop Measurement Flow Control`
	`  retvalStop = mfcStop(false, true)`
	`endif`
	
	`; ckeck again status of ative MFC`
	`status = mfcIsCalculating()`
	`if status = 0 then`
	`  Pause("MFC stopped!")`
	`endif`

mfcStop(stopImmediately, waitUntilStopped)	Stoppt den Measurement Flow Control (MFC). Die beiden Parameter vom Typ boolean haben die folgende Bedeutung: **stopImmediately**: Falls true, wird der MFC sofort beenden. Bei false wird wie beim Stopknopf in der Ribbonbar der Terminate Task ordentlich ausgeführt. **waitUntiStopped**: Bei true blockiert die Formel, bis der MFC beendet wurde, bei false wird umgehend der nächste Schritt in der Fomel ausgeführt.
	
	Rückgabewert **variable** is vom Typ Integer, 0 bedeutet keine Fehler
	
	Example:
	`; create an MFC with two tasks, both enbaled`
	`; with an delay of 5s as executable function`

	``
	; start Measurement Flow Control
	mfcstart()
	
	; do some code
	delay(2)
	
	; stop the second task
	val = mfcStopTask("Task 2")
	if val = 0 then
	  Pause("Task not found!")
	endif
	
	; check for aktive MFC
	status = mfcIsCalculating()
	
	if status = 1 then
	  ; stop Measurement Flow Control
	  retvalStop = mfcStop(false, true)
	endif
	
	; ckeck again status of ative MFC
	status = mfcIsCalculating()
	if status = 0 then
	  Pause("MFC stopped!")
	endif
	``

mfcStopTask(name)	Stoppt einen laufenden MFC Task. Der Parameter name als String entspricht dem Title des entsprechenden Tasks. Rückgabe wert is 1 wenn der Task vorhanden ist. Falls dieser nicht existiern sollte wird 0 zurück gegen. Die Formel bricht in dem Fall aber nicht ab, auch wird keine Exception geworfen.
	
	Example:
	``
	; create an MFC with two tasks, both enbaled
	; with an delay of 5s as executable function
	
	; start Measurement Flow Control
	mfcstart()
	
	; do some code
	delay(2)
	
	; stop the second task
	val = mfcStopTask("Task 2")
	if val = 0 then
	  Pause("Task not found!")
	endif
	
	; check for aktive MFC
	status = mfcIsCalculating()
	
	if status = 1 then
	  ; stop Measurement Flow Control
	  retvalStop = mfcStop(false, true)
	endif
	
	; ckeck again status of ative MFC
	status = mfcIsCalculating()
	if status = 0 then
	  Pause("MFC stopped!")
	endif
	``

34.11 Misc. Functions

Name[Unit] = Expression	Prinzip einer Formel.
	
	Example:
	`; set units to traces`
	`i[A] = c0A1`
	`u[V] = c0A2`
	`P[W] = i * u`
	
	`; modify unit of a trace`
	`UnitP = GetUnit(P)`
	`P2 = P*1000`
	`SetUnit(P, "m"+UnitP)`

Audio_Sr_44100	Liefert den Wert 44100. Diese Konstante kann in der Funktion **PlaySound** verwendet werden um die Standard-Samplerate anzugeben.
	
	Example:
	`tr = Sinus(1,440,0,100E3,100E3)`
	`PlaySound(tr, Audio_Sr_44100)`

BoardDriverVersion	Schlüsselwort zu der Funktion **GetVersion**.
BoardFirmwareVersion	Schlüsselwort zu der Funktion **GetVersion**.
BoardHardwareVersion	Schlüsselwort zu der Funktion **GetVersion**.

ClearReference(a)	Löscht die Einträge und Kurven aus der angegebenen Referenz **a**.
	
	Example:
	`; Clears all existing references`
	`listRef = GetReferenceNames()`
	
	`for each item in listRef`
	`  ClearReference(item)`
	`next`
	
	`; Clear one specific reference`
	`ClearReference("ref1")`

ElapsedWatch()	Liest den internen Zähler, gestartet wird dieser mit der Funktion **StartWatch**. Rückgabewert ist die vergangene Zeit in Millisekunden. Siehe auch die Funktionen **StartWatch** und **StopWatch**.
	
	Example:
	`StartWatch()`
	`Delay(1.1)`
	`t1 = ElapsedWatch() ; approx. 1'100ms`
	`Delay(0.5)`
	`t2 = StopWatch()  ; approx. 1'600ms`

EnableSaveHdf(bool)	Verhindert, dass Dateien im Verzeichnis expr des aktuellen Experiments erstellt werden. Der Parameter **bool** vom Typ Boolean hat die beiden Funktionen: True = *.tpc5 Daten werden generiert, False = keine *tpc5 Dateien werden abgelegt. Standardmässig ist dieser auf True gesetzt.
	
	Example:
	`; Read all files in the current data directory`
	`path = GetPath("data")`
	`files = GetFiles(path, "*.tpc5")`

	``
	; Disable writing of *.tpc5 files
	EnableSaveHdf(False)
	for each item in files
	  trace = File(item,0)
	  minVal = Min(trace)
	  maxVal = Max(trace)
	next
	; Enable writing of *.tpc5 files
	EnableSaveHdf(True)
	
	Pause("values", minVal, maxVal)
	``

Enable-ScreenUpdate(bool)	Aktiviert oder Deaktiviert die Bildschirmaktualisierung während den Berechnungen im Formeleditor. Es wird empfohlen diese zu deaktivieren (**bool** = false) um die Effizienz und Leistung der Formeln zu verbessern. Parameter **bool** ist vom Typ Boolean, True = update aktiv, False = Update ausgeschalten. **Example:** ``` EnableScreenUpdate(false) ; your code here: ; calculation of multiple traces EnableScreenUpdate(True) ```
GetActiveWindow()	Gibt den aktuell selektierten Fensternamen als Text zurück. **Example:** ``` s = GetActiveWindow() Pause(s) ```
GetInfo(key, deviceIdx [, boardIdx [, inputIdx]])	Gibt detaillierte Informationen zu **deviceIdx**, **boardIdx** und **inputIdx** (Gerät, Karte und Kanal) gemäss dem Parameter **key** zurück. Folgende Schlüsselwörter für **key** sind implementiert. **Device:** url, deviceName, deviceId, isSimulated, deviceDescription, deviceMacAddress, useLocalTime, autoStartMeasurement, autoStartAutoSequence, serverPort1, serverPort2, hasTwoInOne, hasTwoInOnePW, dataFilename, nrOfBackupFiles, hdFlushInterval, writeThroughCache, modelType, freeDiskSpace, totalDiskSpace **Board:** hasecr, hasAdvancedTrigger, maxBlockLength, maxSamp-leRate, nrOfInputs, has16Bit, hasSlewRateTrigger, serialNr, boardClass **Input:** hasIcp, hasFilterModule, adcResolution, maxMarkerMask, maxAdcSpeed, hasFullRange, hasDiffModule, has50Ohm, inputClass, isSimulated **Example:** ``` url = GetInfo("URL", 0); IP of device nrOfInputs = GetInfo("nrOfInputs", 0, 0); number of channels of board A ```
GetNrOfBoards(deviceIdx)	Gibt die Anzahl **Boards** (Module) im Gerät **device** an. **deviceIdx = 0** entspricht dem ersten Gerät.

	Example:
	`; change settings for each channel`
	`nrDev = GetNrOfDevices()`
	`for dev = 0 to nrDev-1`
	`  nrBoard = GetNrOfBoards(dev)`
	`  for brd = 0 to nrBoard-1`
	`  nrInp = GetNrOfInputs(dev, brd)`
	`  for inp = 1 to nrInp`
	
	`  ; do some code`
	`  SetRecordingPar(dev,brd,inp,23, 10) ; range 10V`
	
	`  next`
	`  next`
	`next`

GetNrOfDevices()	Gibt die Anzahl der Geräte, die via Sync.Link in einem System sind an. Normalerweise ist nur ein Gerät in einem System. Mehrere Geräte müssen zusätzlich zum Sync.Link via Control Panel -> Devices gemeinsam verbunden werden.
	
	Example:
	`; change settings for each channel`
	`nrDev = GetNrOfDevices()`
	`for dev = 0 to nrDev-1`
	`  nrBoard = GetNrOfBoards(dev)`
	`  for brd = 0 to nrBoard-1`
	`  nrInp = GetNrOfInputs(dev, brd)`
	`  for inp = 1 to nrInp`
	
	`  ; do some code`
	`  SetRecordingPar(dev,brd,inp,23, 10) ; range 10V`
	
	`  next`
	`  next`
	`next`

GetNrOfInputs(deviceIdx, boardIdx)	Gibt die Anzahl **Kanäle** auf einem **board** (Modul) im Gerät **device** an. **deviceIdx = 0** entspricht dem ersten Gerät. **boardIdx = 0** entspricht dem ersten Modul "A" im Gerät.
	
	Example:
	`; change settings for each channel`
	`nrDev = GetNrOfDevices()`
	`for dev = 0 to nrDev-1`
	`  nrBoard = GetNrOfBoards(dev)`
	`  for brd = 0 to nrBoard-1`
	`  nrInp = GetNrOfInputs(dev, brd)`
	`  for inp = 1 to nrInp`
	
	`  ; do some code`
	`  SetRecordingPar(dev,brd,inp,23, 10) ; range 10V`
	
	`  next`
	`  next`
	`next`

GetNrOfReferences()	Gibt die Anzahl Referenzen zurück welche im Signal Source Browser definiert sind.
	
	Example:
	`; Get the number of references from the Signal Source Browser`
	`nRef = GetNrOfReferences()`

GetRecordingPar(deviceIdx, boardIdx, inputIdx, parameter)	Liest die aktuellen Einstellungen eines Hardware Kanals aus. Achtung: **deviceIdx** und **boardIdx** beginnen bei 0, **inputIdx** bei 1.
	
	Paramter ist vom Typ Integer, **string_parameter** ein String. Je nach Befehl wird entweder der eine oder andere Typ verwendet. Eine Liste der verfügbaren Parameter kann in der TpcAccess Dokumentation entnommen werden. http://wiki.elsys-instruments.com/index.php?title=TpcAccess
	
	Example:
	`; read range and channel name of A1`
	`Range = GetRecordingPar(0,0,1, 23)`
	`ChnName = GetRecordingPar(0,0,1, "ChName")`
	
	`; read Operation Mode`
	`; 0=Scope, 1=Multi Block, 2=continuous`
	`; 3=ECR Single Channel, 67=with Dual Mode`
	`; 131=ECR Multi Channel, 195=with Dual Mode`
	`OperationMode = GetRecordingPar(0,0,1, 1)`

GetReference(a)	Liest eine Referenz, einen Kanal, Block oder Marker aus den Referenzen und gibt diesen zurück. Rückgabewert **variable** ist eine Kurve,
	
	Example:
	`; Create a reference`
	`tr1 = Sinus(10,50,0, 1E3,1E3)`
	`tr2 = Noise(2,tr1)`
	`tr3 = tr1 +tr2`
	`tr4 = tr1 -tr2`
	`tr5 = Smooth(tr4, 50)`
	`SetReference("ref1", false, tr1, tr2, tr3, tr4, tr5)`
	
	`trRead = GetReference("ref1",1) ; tr2`

GetReferenceNames()	Gibt eine Liste mit den aktiven Referenzen zurück. Rückgabewert **variable** ist vom Typ List. Siehe auch die Funktion **GetReference** und **NTracesInReference**.
	
	Example:
	`ListRef = GetReferenceNames()  ; get a list with all refe-`
	`rences`
	`ppValList = List()  ; preparea list for peak-`
	`peak values`
	
	`for each item in ListRef  ; go through each reference`
	`  ntr = NTracesInReference(item) ; read the number of traces`
	`in the reference`
	`  for n = 0 to ntr-1 step 1.0  ; to through each trace`
	`  tr = GetReference(item, n)`
	`  ppValList() = PeakPeak(tr) ; Do something the the`
	`traces`
	`  next`
	`next`

GetSSBFiles()	Gibt eine Liste mit allen Dateien aus dem Signal Source Browser zurück. Dateien können per Drag & Drop vom Windows Datei Explorer in den Signal Source Browser gezogen werden. Anschliessend kann die ausgewählte Datei im Formeleditor

	bearbeitet werden. Erlaubte Dateitypen sind *.tpc5, *.tps5, *.tdp and *.bdf für die Analyse im Formeleditor.
	
	Example:
	`fileList = GetSSBFiles()`
	
	`for each item in fileList`
	`  tr = File(item,0)`
	`  ; do something with this trace`
	`next`

GetVersion(key)	Gibt Versionsnummer von Hard- oder Software zurück.
	Je nach verwendetem Schlüsselwort für **key** werden weitere Parameter **deviceIdx** und **boardIdx** benötigt.
	Die Funktion gibt einen String zurück.
	
	0 = ProgramVersion; keine weiteren Parameter additonal parameter
	1 = ServerVersion; deviceIndex
	2 = BoardHardwareVersion; deviceIndex und boardIndex
	3 = BoardDriverVersion; deviceIndex und boardIndex
	4 = BoardFirmwareVersion; deviceIndex und boardIndex
	5 = InputHardwareVersion; deviceIndex und boardIndex\
	
	Example:
	`; create a dictionary`
	`verDict = Dictionary()`
	
	`; read all parameters`
	`verDict("ProgramVersion") = GetVersion(ProgramVersion)`
	`verDict("ServerVersion") = GetVersion(ServerVersion, 0)`
	`verDict("BoardHardwareVersion") = GetVersion(BoardHardwareVer-`
	`sion ,0,0)`
	`verDict("BoardDriverVersion") = GetVersion(BoardDriverVer-`
	`sion,0,0)`
	`verDict("BoardFirmwareVersion") = GetVersion(BoardFirmwareVer-`
	`sion,0,0)`
	`verDict("InputHardwareVersion") = GetVersion(InputHardwareVer-`
	`sion,0,0)`
	
	`; print e.g. TranAX version`
	`pause("TranAX version: "+verDict("ProgramVersion"))`

InputHardwareVersion	Schlüsselwort zu der Funktion **GetVersion**.
	
	Example:
	`; Returns the input hardware version of Board 0A`
	`inputHwVer = GetVersion(InputHardwareVersion, 0, 0)`

MergeTraces([insert-Mark,], trace*) **MergeTraces([insert-Mark,], list)** **MergeTraces(filepath, name, [channelIdx, [blockNumber, trigger-Sample, triggerTime,]] trace*)**	Fügt mehrere Kurven zu einer einzigen zusammen. Wenn der optionale **insertMark** =**True** gesetzt wird, werden in der zusammengefügten Kurve zwischen den einzelnen Originalkurven spezielle **Markierungen** erzeugt. Siehe auch die Funktion **Merge**. Fügt mehrere Kurven aus **list** zu einer Kurve zusammen. Wenn der optionale **insertMark** =**True** gesetzt wird, werden in der zusammengefügten Kurve zwischen den einzelnen

Originalkurven spezielle **Markierungen** erzeugt.
Siehe auch die Funktion **Merge**.

Fügt mehrere Kurven **trace** zu einer Kurve zusammen.
filepath definiert den Pfad und den Dateinamen, in dem die neue Kurve gespeichert wird.
name definiert den Namen des Kurve in der Datei.
Der optionale Parameter **channelIdx** als Zahl, die mit 0 beginnt, definiert den Kanal.
Die optionalen Parameter **blockNumber** definieren einen bestimmten Block, beginnend mit 0, **triggerSample** das Sample für die Triggerposition und **triggerTime** den Zeitstempel des Triggerereignisses.
Bitte beachten Sie auch die Funktion **Merge**.

Example:

```
; merge multiple blocks into one trace
s10 = c0A1.0   ; Block 0 (Multiblock Recording)
s11 = c0A1.1   ; Block 1
s12 = c0A1.2   ; Block 2
s1All = MergeTraces(True, s10, s11, s12)


; create a new signal
tr1 = Sinus(10,50,0,1E3,1E3)
tr2 = Sinus(10,100,0,1E3,1E3)
tr  = MergeTraces(false, tr1, tr2, tr1)


; create a file with a Multi Block recording
trace1 = Sinus(1, 1e3, 0, 1e6, 1e4)
trace2 = Sinus(1, 1e3, 0, 1e6, 1e4)
trace2 = Shift(trace2, 15e-3)
trace2 = SetTrigger(trace2, 15e-3)

filepath = "test.tpc5"
MergeTraces(filepath, "trace", 0, 0, 0, TTrigger(trace1),
trace1)
MergeTraces(filepath, "trace", 0, 1, 0, TTrigger(trace2),
trace2)
CloseFile(filepath)
```

NBlks(trace)
NBlks(filepath, index)

Gibt die Anzahl der Blöcke einer Kurve (**trace**) zurück.

> **trace** ist sinnvollerweise eine Messkurve, welche mit **Multi Block oder ECR Mode** aufgezeichnet wurde.

Gibt die Anzahl der Blöcke einer Kurve (**index**) in einer Datei (**filepath**) zurück.

Example:

```
; get the maximum of all blocks
fname$ = "Test-1.tpc5"
trace = file(fname$,0)
blk = NBlks(trace)
Mx=-1000       ; initialize
i=0
For  i = 0 to blk-1
    value = Max(file(fname$,0).i)
```


<table>
<tr><td></td><td>

```
    If value > Mx Then
        Mx = value
    Endif
Next
```

</td></tr>
<tr><td>NormDist(a, mean, standardDev, cumulativeBool)</td><td>
Gibt die Normalverteilung für den angegebenen Mittelwert und die Standardabweichung zurück. Werte für variable sind 0 ... 1 (0 =0%, 1= 100%).
a: Wert für den die Verteilung gerechnet wird. mean: Das arithmetische Mittel der Verteilung. standardDev: Die Standardabweichung der Verteilung. cumulativeBool: Bestimmt die Form der Funktion. Wenn TRUE ist, gibt NormDist die kumulative Verteilungsfunktion zurück. Wenn es FALSE ist, wird die Wahrscheinlichkeitsmassenfunktion zurückgegeben.

Example:

```
; create some random values
; e.g. time duration for a light barrier in milliseconds
items = 100
myArr = Array(0 to items-1) as double
for a = 0 to items-1 step 1.0
   myArr(a) = random(58, 74)      ; random values form 58ms to
74ms
next

tmp = StdDev(myArr, items)        ; StdDev returns an array
standardDeviation = tmp(items-1)  ; use last item
meanValue = mean(myArr)           ; calculate mean

; calculate the percentage of light barrier values below 68 ms
pcVal = NormDist(68, meanValue, standardDeviation, true)

; calculate the minimal duration to be on the upper 75%
msVal = NormInv(0.25,meanValue,standardDeviation)
```

</td></tr>
<tr><td>NormInv(probability, mean, standardDev)</td><td>
Gibt das Inverse der normalen kumulativen Verteilung für den angegebenen Mittelwert und die Standardabweichung zurück. probability: Eine Wahrscheinlichkeit entsprechend der Normalverteilung. mean: Das arithmetische Mittel der Verteilung. standardDev: Die Standardabweichung der Verteilung.

Example:

```
; create some random values
; e.g. time duration for a light barrier in milliseconds
items = 100
myArr = Array(0 to items-1) as double
for a = 0 to items-1 step 1.0
   myArr(a) = random(58, 74)      ; random values form 58ms to
74ms
next

tmp = StdDev(myArr, items)        ; StdDev returns an array
standardDeviation = tmp(items-1)  ; use last item
meanValue = mean(myArr)           ; calculate mean

; calculate the percentage of light barrier values below 68 ms
pcVal = NormDist(68, meanValue, standardDeviation, true)

; calculate the minimal duration to be on the upper 75%
msVal = NormInv(0.25,meanValue,standardDeviation)
```

</td></tr>
<tr><td>NTracesInReference(a)</td><td>Gibt die Anzahl der Kurven in einer Referenz a zurück, Rückgabewert ist eine Zahl.</td></tr>
</table>

	Falls eine Referenz leer ist oder nicht existiert, gibt diese Funktion 0 zurück, ansonsten die Anzahl der vorhandenen Kurven.
	
	Siehe auch die Funktionen **GetReference** und **GetReferenceNames**.
	
	Example:
	`; simple example`
	`ntr = NTracesInReference("ref12")`
	
	`; more advanced example`
	`ListRef = GetReferenceNames()  ; get a list with all refe-`
	`rences`
	`ppValList = List()  ; preparea list for peak-`
	`peak values`
	
	`for each item in ListRef  ; go through each reference`
	`  ntr = NTracesInReference(item) ; read the number of traces`
	`in the reference`
	`  for n = 0 to ntr-1 step 1.0  ; to through each trace`
	`  tr = GetReference(item, n)`
	`  ppValList() = PeakPeak(tr) ; Do something the the`
	`traces`
	`  next`
	`next`

Pi	Die Keiszahl Pi = 3.14159265359.
	
	Example:
	`; RC lowpass`
	`R = 1E3`
	`C = 100E-9`
	`fg = 1/(2*Pi*R*C) ; fg = approx. 1.6kHz`
	
	`; area of a disk`
	`r = 0.5`
	`A = r^2 * pi`

PlaySound(a [, fs])	Audio- und tpc5 Dateien können über den Audio-Ausgang des Computers abgespielt werden. Unterstützt werden die Formate *.wav, *.mp3 und *.tpc5. Es können auch berechnete Kurven **trace** anstelle einer Datei abgespielt werden.
	
	Bei *.tpc5 Dateien bzw. **a** muss der Parameter **fs** (Samplerate in Hz) angegeben werden. Die maximale Samplerate, wie auch die maximale Amplitude, hängt vom installierten Audio System ab. D.h. ein Abspielen direkt von einer *.tpc5 Datei ist nur dann gewährleistet, wenn die Kurve auf der Datei eine entsprechende Abtastrate aufweist. Es empfiehlt sich, durch ein vorangehendes Resampling mit Skalierung die Kurve auf eine geeignete Audio-Frequenz und Amplitude zu konvertieren, siehe Beispiel.
	
	Bei *.wav und *.mp3 fehlt der Parameter **fs**, diese Information wird direkt aus der Datei gelesen.
	
	Example:
	`AmpFact=0.5`
	`; Left channel`
	`sLeft=File("SoundData.tpc5",0)`
	`; Right channel`
	`sRight=File("SoundData.tpc5",1)`

	`; Adapt to audio frequency`
	`sLeft=AmpFact*Resampling(sLeft, 44100)`
	`sRight=AmpFact*Resampling(sRight, 44100)`
	`; Left and right channel in one file`
	`Save ("SoundStereo.tpc5", sLeft, sRight)`
	`srRec=1/TSample(sLeft)`
	`; Play it`
	`PlaySound("SoundStereo.tpc5", srRec)`

ProgramVersion	Schlüsselwort zu der Funktion **GetVersion**.
	
	Example:
	`; Returns the version of the application`
	`progVer = GetVersion(ProgramVersion)`

Random(min, max)	Gibt eine Zufallszahl zwischen den definierten Grenzen **min** und **max** zurück.
	
	Example:
	`; one random number`
	`n = Random(0,100)`
	
	`; create an array with random numbers`
	`myArr = Array(0 to 9) as double`
	`for a = 0 to Length(myArr)-1 step 1.0`
	`  myArr(a) = Random(-10, 10)`
	`next`

ServerVersion	Schlüsselwort zu der Funktion **GetVersion**.
	
	Example:
	`; Returns the version of the server of device 0`
	`progVer = GetVersion(ServerVersion, 0)`

SetFormulaError(text)	Die Funktion kann z.B. dazu verwendet werden um an einer bestimmten Stelle eine Fehlermeldung abzusetzen und dadurch zu kennzeichnen, dass weitere Funktionen nicht mehr gültige Resultate liefern werden.
	
	Example:
	`Tb=TBegin(trc)  ; initialize`
	`for a = 1 to 10`
	`  Tcr = TCross(trc,tb,TEnd(trc),0)`
	`  tb = Tcr  ; for next`
	`  If Tcr = NotDefined Then`
	`  text$ = "Missing Zero Crossings"`
	`  SetFormulaError(text$)`
	`  endif`
	`next`

SetRecordingMode(deviceIdx, mode)	Setzt den Aufnahme Modus des Gerätes **deviceIdx**. Der Parameter **deviceIdx** ist vom Typ Integer und in den meisten Fällen 0. Bei mehreren Geräten mit Synchronisation ist dieser entsprechend der Einbindung in TranAX, in der Regel der IP Adresse aufsteigend gegliedert.
	
	Der Aufnahme Modus **mode** vom Typ Integer kann die folgenden Zustände haben: Scope = 0, MultiBlock = 1, Continuous = 3, EcrSingle = 4, EcrMulti = 5, EcrSingleDual = 6, EcrMultiDual = 7
	
	Example:
	`; continuous recording`
	`SetRecordingMode(0, 3)`

<table>
<tr>
<td>SetRecordingPar(deviceIdx, boardIdx, inputIdx, parameter, value)</td>
<td>Setzt die Einstellungen eines Hardware Kanals.

Achtung: device und board beginnen bei 0, input bei 1.

value ist vom Typ Zahl oder Text. Je nach Parameter wird der eine oder andere Typ verwendet. Eine Liste der verfügbaren Parameter kann in der TpcAccess Dokumentation gefunden werden: Elsys-Wiki

Je nach parameter wird eine Zahl oder ein Text erwatet für value.

Example:
<pre>
; set range of A1 to 10V
SetRecordingPar(0, 0, 1, 23, 10)
; set channel name A1
SetRecordingPar(0, 0, 1, "ChName", "Input Voltage")
; set offset 75%
SetRecordingPar(0, 0, 1, 24, 75) ;
; set input mode "Single Ended"
SetRecordingPar(0, 0, 1, 20, 1) ; 0:Off, 1:SE, 2:Diff

endformula
; change settings for each channel
nrDev = GetNrOfDevices()
for dev = 0 to nrDev-1
  nrBoard = GetNrOfBoards(dev)
  for brd = 0 to nrBoard-1
    nrInp = GetNrOfInputs(dev, brd)
    for inp = 1 to nrInp

        ; do some code
        SetRecordingPar(dev,brd,inp,23, 10) ; range 10V

    next
  next
next
</pre></td>
</tr>
<tr>
<td>SetReference(name, fileList*)
SetReference(name, assignHardware, trace*)</td>
<td>Weist einer Referenz Hardwarekanäle oder Kurven zu. Referenzen können dazu verwendet werden, Messkurven in einem definierten Layout schnell und ohne Aufwand zu ersetzen. Siehe dazu auch die Möglichkeiten des Signal Source Browsers. Parameter dazu sind:
name: Ist die Referenz selbst als String ("ref1", "ref2", … "refn")
fileList: Ist vom Typ List(), dient der Zuweisung von Kurven aus Dateien im *.tpc5 oder *.bdf Format

Weist einer Referenz Hardwarekanäle oder Kurven zu. Referenzen können dazu verwendet werden, Messkurven in einem definierten Layout schnell und ohne Aufwand zu ersetzen. Siehe dazu auch die Möglichkeiten des Signal Source Browsers. Parameter dazu sind:
name: Ist die Referenz selbst als String ("ref1", "ref2", … "refn")
assignHardware: Dient der Zuweisung eines Hardware Kanals, Parameter ist vom Typ Boolean. Falls ein Hardwarekanal über eine Variable zugewiesen ist, wird bei True der Kanalname, bei</td>
</tr>
</table>

<table>
<tr><td></td><td>False der Variablenname in den Referenzen gelistet. Bei einer direkten Zuweisung eines Hardwarekanals (c0A1, …) bei den Funktionsparametern, muss True verwendet werden.
trace*: Sind Kanäle, Variablen oder berechnete Kurven aus Formeln

Example:

```
; Add hardware channels to a reference
; Hardware name (c0A1, ...) will be listed in the references
SetReference("ref1", true, c0A1, c0A2)

; Add hardware channels as a variable to a reference
; Hardware names (c0A1, ...) will be listed in the references
tr1 = c0A1
tr2 = c0A2
SetReference("ref2", true, tr1, tr2)

; Add hardware channels as a variable to a reference
; Variable names will be listed in the references
tr1 = c0A1
tr2 = c0A2
SetReference("ref3", false, tr1, tr2)

; Mixed application, traces and hardware channels
; Variable names will be listed in the references
measured = c0A1
calculated = Sinus(10,50,0, measured)
delta = measured - calculated
filtered = Smooth(delta,500)
SetReference("ref4", false, measured, calculated, delta, fil-
tered)

; Using files
fname = "mytrace.tpc5"
Save(fname, calculated, filtered)
listFname = List(fname, 1) ; Either a trace number in the file
(starts with 0) or -1 for all files
SetReference("ref5", listFname)
```
</td></tr>
<tr><td>**StartWatch()**</td><td>Startet den internen Zähler, dieser kann zum Debuggen des Formelcodes, messen von Laufzeiten verwendet werden. Siehe auch die Funktionen **ElapsedWatch** und **StopWatch**.

Example:

```
StartWatch()
Delay(1.1)
t1 = ElapsedWatch() ; approx. 1'100ms
Delay(0.5)
t2 = StopWatch()  ; approx. 1'600ms
```
</td></tr>
<tr><td>**StopWatch()**</td><td>Stoppt den internen Zähler, gestartet wird dieser mit der Funktion **StartWatch**. Rückgabewert ist die vergangene Zeit in Millisekunden. Siehe auch die Funktionen **StartWatch** und **ElapsedWatch**.

Example:

```
StartWatch()
Delay(1.1)
t1 = ElapsedWatch() ; approx. 1'100ms
Delay(0.5)
t2 = StopWatch()  ; approx. 1'600ms
```
</td></tr>
<tr><td>**UseMemory(bool)**</td><td>Falls **bool = True** ist, werden Kurven-Resultate im Arbeitsspeicher und nicht als Datei im Verzeichnis abgelegt.</td></tr>
</table>

	Dies kann die Berechnungszeit erheblich verkürzen. **UseMemory(bool)** entspricht der Funktion **SetEnvironment(0, true).** **Example:** `UseMemory(True) ; Curve data is stored in RAM`

34.12 PLC Functions

plcAnalogRead(connectionAddr, pin [,mode])	Definiert einen Pin als analogen Eingang. Bitte beachten Sie, dass nicht jeder Pin als analoger Eingang konfiguriert werden kann. Rückgabewert **variable** ist der ADC-Wert selbst, für Elsys-PLC ein Wert zwischen 0 und 1023 (10-Bit-ADC). **connectionAddr**: Serielle Schnittstelle oder IP-Adresse der Elsys PLC als String. Z.B. "COM12" für eine serielle Schnittstelle oder "192.168.0.35" für eine IP-Adresse **pin**: Arduino Pin entsprechend der Pin-Out-Liste **Modus**: Mit dem optionalen Parameter **modus** werden Statistikdaten vom Analogeingang gelesen. Das Aktualisierungsintervall ist auf 100 ms gesetzt. Mögliche Werte für den Parameter **Modus** sind: 0 (oder nicht definiert): Aktueller ADC-Wert 1: Minimalwert 2: Maximalwert 3: Mittelwert 4: Effektivwert 666: Statistikdaten für Modus 1-4 oben zurücksetzen
	
	Example:
	`; Input AI0 used as an analogue input`
	`WriteForNext("PORT","192.168.0.35")`
	`port = ReadOfPrevious("PORT")`
	`plcOpen(port)`
	
	`; Clear Analogue Statistic`
	`plcAnalogRead(port, 54, 666)`
	
	`adcval  = plcAnalogRead(port, 54) ; equal to plcAnalog-Read(port, 54, 0)`
	`minval  = plcAnalogRead(port, 54, 1)`
	`maxval  = plcAnalogRead(port, 54, 2)`
	`meanval = plcAnalogRead(port, 54, 3)`
	`rmsval  = plcAnalogRead(port, 54, 4)`
	
	`plcClose(port)`

plcAnalogWrite(connectionAddr, pin, value [, duration] [, mode])	Definiert einen Pin als Analogausgang. Bitte beachten Sie, dass nicht jeder Pin als Analogausgang konfiguriert werden kann, nur Pins welche PWM unterstützen. **connectionAddr**: Serielle Schnittstelle oder IP-Adresse der Elsys PLC als String. Z.B. "COM12" für eine serielle Schnittstelle oder "192.168.0.35" für eine IP-Adresse **pin**: Arduino Pin entsprechend der Pin-Out-Liste **value**: Definiert das Ausgangssignal in tausendstel (0,1%) Intervallen. 0 bedeutet 0V, 1000 bedeutet maximales Ausgangssignal, 10V für Elsys PLC. **duration**: Mit dem optionalen Parameter **duration** wird eine Rampe als Ausgangssignal erzeugt. Nach Ablauf der Zeit in Sekunden erreicht

	es den Analogausgangswert. **mode**: Mit dem optionalen Parameter **mode** wird ein exponentielles Ausgangssignal erzeugt, das das lineare Dimmen von angeschlossenen LEDs ermöglicht. Nicht definiert, False oder 0 bedeutet lineare Ausgabe, True oder 1 bedeutet exponentielle Ausgabe
	
	Example:
	`; Example 1, Ouput AO0 used as an analogue output.`
	`WriteForNext("PORT","192.168.0.35")`
	`port = ReadOfPrevious("PORT")`
	`plcOpen(port)`
	
	`; Set output to 0V`
	`plcAnalogWrite(port, 12, 0)`
	`; Wait 1 second`
	`delay(1)`
	`; Set output to 10V, ramp for 2 seconds`
	`plcAnalogWrite(port, 12, 1000, 2)`
	
	`plcClose(port)`
	
	`; Example 2, Ouput AO0 used as an analogue output for LEDs.`
	`WriteForNext("PORT","192.168.0.35")`
	`port = ReadOfPrevious("PORT")`
	`plcOpen(port)`
	
	`; Set output to 0V`
	`plcAnalogWrite(port, 12, 0)`
	`; Wait 1 second`
	`delay(1)`
	`; Set output to 10V, ramp for 2 seconds`
	`plcAnalogWrite(port, 12, 1000, 2, True)`
	
	`plcClose(port)`

plcClose(connectionAddr)	Schliesst eine serielle Schnittstelle oder eine Ethernetverbindung zu einem Elsys PLC. **connectionAddr**: Serielle Schnittstelle oder IP-Adresse der Elsys PLC als String. Z.B. "COM12" für eine serielle Schnittstelle oder "192.168.0.35" für eine IP-Adresse
	
	Example:
	`; Examples 1, default Ethernet connection:`
	`; open a Ethernet Connection`
	`WriteForNext("PORT","192.168.0.35")`
	`port = ReadOfPrevious("PORT")`
	`plcOpen(port)`
	`; your code here`
	`plcClose(port)`
	
	`; Examples 2, default serial port connection:`
	`; open a Serial Port Connection`
	`WriteForNext("PORT","COM15")`
	`port = ReadOfPrevious("PORT")`
	`plcOpen(port)`
	`; your code here`
	`plcClose(port)`
	
	`; Examples 3, custom serial port connection:`
	`dictParams = Dictionary()`
	`dictParams("baudrate") = 115200`
	`dictParams("readTimeout") = 2000`
	`dictParams("writeTimeout") = 2000`
	`dictParams("newLine") = "\n"`
	`dictParams("dtrEnable") = False`

	`WriteForNext("PORT","COM15")`
	`port = ReadOfPrevious("PORT")`
	`plcOpen(port, dictParams)`
	`; your code here`
	`plcClose(port)`

plcDigitalRead(connectionAddr, pin [,overwrite])	Definiert einen Pin als digitalen Eingang. Der Rückgabewert ist 0, wenn das Eingangssignal LOW, und 1, wenn das Eingangssignal HIGH ist.
	
	connectionAddr: Serielle Schnittstelle oder IP-Adresse der Elsys PLC als String. Z.B. "COM12" für eine serielle Schnittstelle oder "192.168.0.35" für eine IP-Adresse **pin**: Arduino Pin entsprechend der Pin-Out-Liste **overwrite**: Der optionale Parameter **overwrite** wird verwendet um eine Pin-Funktion neu zu definieren. Wenn ein Pin als Ausgang definiert ist, gibt plcDigitalRead seinen Wert zurück. Mit dem optionalen Parameter overwrite = true wird der Pin als Digitaleingang neu konfiguriert.
	
	Example:
	`; Input AI0 is connection with a push button to Vin of the Elsys PLC.`
	`WriteForNext("PORT","192.168.0.35")`
	`port = ReadOfPrevious("PORT")`
	`plcOpen(port)`
	
	`; Pushbutton pressed InputVal= 1, else = 0`
	`InputVal = plcDigitalRead(port, 54)`
	
	`plcClose(port)`

plcDigitalWrite(connectionAddr, pin, value [, pulseDuration])	Definiert einen Pin als digitalen Ausgang und setzt seinen Status..
	
	connectionAddr: Serielle Schnittstelle oder IP-Adresse der Elsys PLC als String. Z.B. "COM12" für eine serielle Schnittstelle oder "192.168.0.35" für eine IP-Adresse **pin**: Arduino Pin entsprechend der Pin-Out-Liste **value**: Gültige Parameter für sind: **plcLow**, gleich dem ganzzahligen Wert 0, der Ausgang ist OFF, **plcHigh**, gleich dem ganzzahligen Wert 1, Ausgang ist ON, **plcToggle**, gleich dem ganzzahligen Wert 2, invertiert den Ausgang (ON schaltet nach OFF und umgekehrt). Alle drei Parameter sind vordefinierte Schlüsselwörter. **pulseDuduration**; Mit dem optionalen Parameter **pulseDuration** wird eine Monoflop-Funktion definiert. Der Ausgang wird für die definierte Dauer in Sekunden (Integer oder Float) auf den Wert **value** gesetzt.
	
	Example:
	`; Example 1: Two examples of a Monoflop Output.`
	`; The second solution with additional parameter`
	`; pulseDuduration doesn't block the code execution`
	`WriteForNext("PORT","192.168.0.35")`
	`port = ReadOfPrevious("PORT")`
	`plcOpen(port)`
	
	`; 1. Monoflop`
	`plcDigitalWrite(port,22,plcHigh)`
	`Delay(1.5)`
	`plcDigitalWrite(port,22,plcLow)`

	``` ; 2. Monoflop plcDigitalWrite(port,22,plcHigh, 1.5) plcClose(port) ;Example 2: Toggle output from ON to OFF and vice versa. WriteForNext("PORT","192.168.0.35") port = ReadOfPrevious("PORT") plcOpen(port) plcDigitalWrite(port,22,plcToggle) plcClose(port) ```
plcHigh	Schlüsselwort **value** in Funktion **plcDigitalWrite** **plcLow**, gleich dem ganzzahligen Wert 0, der Ausgang ist OFF **plcHigh**, gleich dem ganzzahligen Wert 1, Ausgang ist ON **plcToggle**, gleich dem ganzzahligen Wert 2, invertiert den Ausgang (ON schaltet nach OFF und umgekehrt) **Example:** ``` ; Example 1: Two examples of a Monoflop Output. ; The second solution with additional parameter ; pulseDuduration doesn't block the code execution WriteForNext("PORT","192.168.0.35") port = ReadOfPrevious("PORT") plcOpen(port) ; 1. Monoflop plcDigitalWrite(port,22,plcHigh) Delay(1.5) plcDigitalWrite(port,22,plcLow) ; 2. Monoflop plcDigitalWrite(port,22,plcHigh, 1.5) plcClose(port) ;Example 2: Toggle output from ON to OFF and vice versa. WriteForNext("PORT","192.168.0.35") port = ReadOfPrevious("PORT") plcOpen(port) plcDigitalWrite(port,22,plcToggle) plcClose(port) ```
plcLow	Schlüsselwort **value** in Funktion **plcDigitalWrite** **plcLow**, gleich dem ganzzahligen Wert 0, der Ausgang ist OFF **plcHigh**, gleich dem ganzzahligen Wert 1, Ausgang ist ON **plcToggle**, gleich dem ganzzahligen Wert 2, invertiert den Ausgang (ON schaltet nach OFF und umgekehrt) **Example:** ``` ; Example 1: Two examples of a Monoflop Output. ; The second solution with additional parameter ; pulseDuduration doesn't block the code execution WriteForNext("PORT","192.168.0.35") port = ReadOfPrevious("PORT") plcOpen(port) ; 1. Monoflop plcDigitalWrite(port,22,plcHigh) Delay(1.5) ```

```
plcDigitalWrite(port,22,plcLow)

; 2. Monoflop
plcDigitalWrite(port,22,plcHigh, 1.5)

plcClose(port)

;Example 2: Toggle output from ON to OFF and vice versa.
WriteForNext("PORT","192.168.0.35")
port = ReadOfPrevious("PORT")
plcOpen(port)

plcDigitalWrite(port,22,plcToggle)

plcClose(port)
```

plcOpen(connectionAddr [, parameters])

Öffnet eine serielle Schnittstelle oder Ethernet-Verbindung (Port 80) zu einem Elsys PLC. Es ist möglich mehrere Geräte zu kombinieren. Daher benötigt jede SPS-Funktion und jedes Kommando den Parameter **connectionAddr**. Es wird empfohlen die Verbindungsadresse als Variable zu speichern. Bei Änderung der IP-Adresse oder der Nummer der seriellen Schnittstelle müssen Änderungen nur einmal im Code vorgenommen werden. In den Beispielen wird **conncetionAddr** mit der Funktion **WriteForNext** gespeichert und mit **ReadOfPrevious** gelesen. Der Vorteil dieser beiden Funktionen ist die Verwendung von **connectionAddr** in Codesegmenten der Measurement Flow Control (MFC) und in Funktionen innerhalb des Formel-Editor-Codes.

Bei einer noch offenen Verbindung zu Beginn einer Formel versucht **plcOpen** automatisch, den seriellen Port oder die Verbindung aus der vorherigen Berechnung wieder zu verwenden. Wenn ein serielles Gerät bereits von einer anderen Anwendung geöffnet wurde, muss diese Anwendung zuerst geschlossen werden.

connectionAddr: Serielle Schnittstelle oder IP-Adresse der Elsys PLC als String. Z.B. "COM12" für eine serielle Schnittstelle oder "192.168.0.35" für eine IP-Adresse **parameter**: Im Allgemeinen sind keine zusätzlichen Parameter erforderlich. Bei der Neudefinition der Parameter der seriellen Schnittstelle ist ein Wörterbuch mit folgenden Parametern zulässig: **baudrate**, **dataBits**, **stopBits**, **readTimeout**, **writeTimeout**, **newline** and **dtrEnable**

Example:

```
; Examples 1, default Ethernet connection:
; open a Ethernet Connection
WriteForNext("PORT","192.168.0.35")
port = ReadOfPrevious("PORT")
plcOpen(port)
; your code here
plcClose(port)

; Examples 2, default serial port connection:
; open a Serial Port Connection
WriteForNext("PORT","COM15")
port = ReadOfPrevious("PORT")
plcOpen(port)
; your code here
```


	`plcClose(port)`
	
	`; Examples 3, custom serial port connection:`
	`dictParams = Dictionary()`
	`dictParams("baudrate") = 115200`
	`dictParams("readTimeout") = 2000`
	`dictParams("writeTimeout") = 2000`
	`dictParams("newLine") = "\n"`
	`dictParams("dtrEnable") = False`
	
	`WriteForNext("PORT","COM15")`
	`port = ReadOfPrevious("PORT")`
	`plcOpen(port, dictParams)`
	`; your code here`
	`plcClose(port)`

plcRead (connectionAddr [, delayMs [, collectResults]])	Liest einen Parameter von **connectionAddr**. Gibt abhängig von der Antwort einen String, eine Zahl oder eine Liste zurück. Wenn ein Timeout auftritt, ist der Rückgabewert NotDefined.
	
	connectionAddr: Serielle Schnittstelle oder IP-Adresse der Elsys PLC als String. Z.B. "COM12" für eine serielle Schnittstelle oder "192.168.0.35" für eine IP-Adresse **delayMs**: Definiert eine Verzögerung in Millisekunden nach dem Lesen der Daten. Dies kann verwendet werden, wenn der nächste Lese- oder Schreibbefehl eine bestimmte Zeit warten muss. **collectResults**: Wenn Sie mehrere Lesevorgänge durchführen möchten, setzen Sie diesen Wert auf true. Sie erhalten eine Liste mit allen Rückgabewerten.
	
	Example:
	`WriteForNext("PORT","COM12")`
	`port = ReadOfPrevious("PORT")`
	`plcOpen(port)`
	
	`; read data from a 3rd party serial device`
	`val = plcRead(port, 200, true)`
	
	`plcClose(port)`

plcTemperatureRead(connectionAddr, pin [, sensor])	Definiert einen Pin als OneWire-Pin für DS1820-Temperatursensoren.
	
	connectionAddr: Serielle Schnittstelle oder IP-Adresse der Elsys PLC als String. Z.B. "COM12" für eine serielle Schnittstelle oder "192.168.0.35" für eine IP-Adresse **sensor**: Mit dem optionalen Parameter **sensor** wird die Nummer eines angeschlossenen Sensors festgelegt. OneWire unterstützt bis zu 127 Sensoren. Nicht definiert bedeutet der erste Sensor in der Kette, 0… 126 eine bestimmte Sensornummer, 666 die Anzahl der Sensorenin der OneWire-Kette.
	
	Rückgabewert **variable** ist dann entweder die Temperur des Sensors oder die Anzahl der vorhanden Sensoren.
	
	Example:
	`; OneWire on Pin A1, 55, with two DS18B20 sensors, 4k7 Ohm Resistor Pullup`
	`WriteForNext("PORT","192.168.0.35")`

	`port = ReadOfPrevious("PORT")`
	`plcOpen(port)`
	
	`; Get Number of sensors`
	`TempSensors = plcTemperatureRead(port, 55, 666)`
	`; Read first sensor`
	`val1 = plcTemperatureRead(port, 55)`
	`; Read second sensor`
	`val2 = plcTemperatureRead(port, 55, 1)`
	
	`plcClose(port)`

plcToggle	Schlüsselwort **value** in Funktion **plcDigitalWrite**
	
	plcLow, gleich dem ganzzahligen Wert 0, der Ausgang ist OFF **plcHigh**, gleich dem ganzzahligen Wert 1, Ausgang ist ON **plcToggle**, gleich dem ganzzahligen Wert 2, invertiert den Ausgang (ON schaltet nach OFF und umgekehrt)
	
	Example:
	`; Example 1: Two examples of a Monoflop Output.`
	`; The second solution with additional parameter`
	`; pulseDuduration doesn't block the code execution`
	`WriteForNext("PORT","192.168.0.35")`
	`port = ReadOfPrevious("PORT")`
	`plcOpen(port)`
	
	`; 1. Monoflop`
	`plcDigitalWrite(port,22,plcHigh)`
	`Delay(1.5)`
	`plcDigitalWrite(port,22,plcLow)`
	
	`; 2. Monoflop`
	`plcDigitalWrite(port,22,plcHigh, 1.5)`
	
	`plcClose(port)`
	
	`;Example 2: Toggle output from ON to OFF and vice versa.`
	`WriteForNext("PORT","192.168.0.35")`
	`port = ReadOfPrevious("PORT")`
	`plcOpen(port)`
	
	`plcDigitalWrite(port,22,plcToggle)`
	
	`plcClose(port)`

plcWrite(connectionAddr, message [, delayMs [, collectResults]])	Schreiben Sie eine Nachricht basierend auf einer Zeichenfolge an **connectionAddr**. Wenn ein Timeout auftritt, ist der Rückgabewert NotDefined. **connectionAddr**: Serielle Schnittstelle oder IP-Adresse der Elsys PLC als String. Z.B. "COM12" für eine serielle Schnittstelle oder "192.168.0.35" für eine IP-Adresse **message**: Elsys PLC spezifischer String, für andere Geräte gemäss deren Spezifikation. **delayMs**: Definiert eine Verzögerung in Millisekunden nach dem Schreiben der Daten. Dies kann verwendet werden, wenn der nächste Lese- oder Schreibbefehl eine bestimmte Zeit warten muss. **collectResults** Setzen Sie collectResults auf true, wenn Sie alle Antworten automatisch erfassen möchten. Sie erhalten eine Liste mit Werten. **Example:** `; Example 1: Read an analogue input`


```
WriteForNext("PORT","192.168.0.35")
port = ReadOfPrevious("PORT")
plcOpen(port)

; read analog input A0 RMS value
val = plcWrite(port, "PN:54;TP:3;P0:4;")

plcClose(port)



; Example 2: Control an Elsys AE-Amp device
dictParams = Dictionary()
dictParams("baudrate") = 19200
dictParams("readTimeout") = 2000
dictParams("writeTimeout") = 2000
dictParams("newLine") = "\n"
dictParams("dtrEnable") = False

WriteForNext("PORT","COM3")
port = ReadOfPrevious("PORT")
plcClose(port)
plcOpen(port, dictParams)

plcWrite(port, "RESET;")
Delay(0.2)
plcWrite(port, "MUX:0;SETHVVAL:400;")
Delay(0.2)
pause("Check High Voltage Signal!");
plcWrite(port, "MUX:0;SETHVPULSE:60;")
Delay(0.2)
plcWrite(port, "MUX:0;SETHVINR:5;")
Delay(0.2)
plcWrite(port, "ADD:0;CHN:1;SETHV:1;")
Delay(0.2)
plcWrite(port, "MUX:0;PULSESTART:666;")
Delay(0.1)

plcClose(port)
```

34.13 Power Functions

ApparentPower(a, b [, start, end [, level, hysteresis]])

Berechnet die Scheinleistung aus **a** und **b**. Ein Parameter sollte die Spannung, der andere der Strom sein. Der Rückgabewert ist Skalar vom Typ double. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll. Weitere optionale Parameter sind **level** und **hysteresis**. Sind diese nicht definiert, wird **level** auf den Mittelwert des Spitzenwerts gelegt, **hysteresis** beträgt 5% des Spitzenwerts.

Example:

```
; Generate two test traces, phase shift of current
deltaPhase = 20 ; would be 20° which indicates an inductive
load
voltageTrace = Sinus(330, 50, 0, 1e6, 1e6)
currentTrace = Sinus(10, 50, -1*deltaPhase, 1e6, 1e6)

; 1.) Formula Editor power functions
cosphi_1 = CosPhi(voltageTrace, currentTrace)
ApparentPower_1[VA] = ApparentPower(voltageTrace, currentTrace)
RealPower_1[W] =  RealPower(voltageTrace, currentTrace)
ReactivePower_1[VAR] = ReactivePower(voltageTrace, current-
Trace)
crestfac_1 = CrestFactorPeriodic(voltageTrace)
powerfac_1 = PowerFactor(voltageTrace, currentTrace)
```

```
; 2.) Manual calculation
phaseTrace = Phase(voltageTrace, currentTrace)
phaseShift_2[°] = phaseTrace(TEnd(phaseTrace))
rmsU[V] = RMSP(voltageTrace)
rmsI[A] = RMSP(currentTrace)

cosphi_2 = cos(phaseShift_2)
ApparentPower_2[VA] = rmsU* rmsI
RealPower_2[W] =  rmsU* rmsI * cos(phaseShift_2)
ReactivePower_2[VAR] = rmsU* rmsI * sin(phaseShift_2)
crestfac_2 = max(voltageTrace) / rmsu
powerfac_2 = cos(phaseShift_2) ; the same as cosPhi


; 3.) some more information
; cosPhi = RealPower / ApparentPower
; cosPhi = PowerFactor
; cosPhi = R / Z  = resistance / impedance = Wirkwiderstand /
Scheinwiderstand
```

CosPhi(a, b [, start, end [, level, hysteresis]])

Berechnet den Cosinus Phi aus **a** und **b**. Der Rückgabewert ist ein Skalar vom Typ Double. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll. Weitere optionale Parameter sind **level** und **hysteresis**. Sind diese nicht definiert, wird **level** auf den Mittelwert des Spitzenwerts gelegt, **hysteresis** beträgt 5% des Spitzenwerts.

> Um sinnvolle Resultate zu erreichen, sollten min. 3.5 Perioden vorhanden sein.

Example:

```
; Generate two test traces, phase shift of current
deltaPhase = 20 ; would be 20° which indicates an inductive
load
voltageTrace = Sinus(330, 50, 0, 1e6, 1e6)
currentTrace = Sinus(10, 50, -1*deltaPhase, 1e6, 1e6)

; 1.) Formula Editor power functions
cosphi_1 = CosPhi(voltageTrace, currentTrace)
ApparentPower_1[VA] = ApparentPower(voltageTrace, currentTrace)
RealPower_1[W] =  RealPower(voltageTrace, currentTrace)
ReactivePower_1[VAR] = ReactivePower(voltageTrace, current-
Trace)
crestfac_1 = CrestFactorPeriodic(voltageTrace)
powerfac_1 = PowerFactor(voltageTrace, currentTrace)


; 2.) Manual calculation
phaseTrace = Phase(voltageTrace, currentTrace)
phaseShift_2[°] = phaseTrace(TEnd(phaseTrace))
rmsU[V] = RMSP(voltageTrace)
rmsI[A] = RMSP(currentTrace)

cosphi_2 = cos(phaseShift_2)
ApparentPower_2[VA] = rmsU* rmsI
RealPower_2[W] =  rmsU* rmsI * cos(phaseShift_2)
ReactivePower_2[VAR] = rmsU* rmsI * sin(phaseShift_2)
crestfac_2 = max(voltageTrace) / rmsu
powerfac_2 = cos(phaseShift_2) ; the same as cosPhi


; 3.) some more information
; cosPhi = RealPower / ApparentPower
; cosPhi = PowerFactor
```


	`; cosPhi = R / Z  = resistance / impedance = Wirkwiderstand / Scheinwiderstand`

CrestFactor(a [, start, end])	Berechnet den Scheitelfaktor (Crest-Faktor) von **a**. Dieser beschreibt das Verhältnis vom absoluten Spitzenwert zu RMS des Parameters **a**. Der Rückgabewert ist ein Skalar vom Typ Double. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll.
	
	Example:
	`; Generate two test traces, phase shift of current`
	`deltaPhase = 20 ; would be 20° which indicates an inductive load`
	`voltageTrace = Sinus(330, 50, 0, 1e6, 1e6)`
	`currentTrace = Sinus(10, 50, -1*deltaPhase, 1e6, 1e6)`
	
	`; 1.) Formula Editor power functions`
	`cosphi_1 = CosPhi(voltageTrace, currentTrace)`
	`ApparentPower_1[VA] = ApparentPower(voltageTrace, currentTrace)`
	`RealPower_1[W] =  RealPower(voltageTrace, currentTrace)`
	`ReactivePower_1[VAR] = ReactivePower(voltageTrace, current-Trace)`
	`crestfac_1 = CrestFactorPeriodic(voltageTrace)`
	`powerfac_1 = PowerFactor(voltageTrace, currentTrace)`
	
	`; 2.) Manual calculation`
	`phaseTrace = Phase(voltageTrace, currentTrace)`
	`phaseShift_2[°] = phaseTrace(TEnd(phaseTrace))`
	`rmsU[V] = RMSP(voltageTrace)`
	`rmsI[A] = RMSP(currentTrace)`
	
	`cosphi_2 = cos(phaseShift_2)`
	`ApparentPower_2[VA] = rmsU* rmsI`
	`RealPower_2[W] =  rmsU* rmsI * cos(phaseShift_2)`
	`ReactivePower_2[VAR] = rmsU* rmsI * sin(phaseShift_2)`
	`crestfac_2 = max(voltageTrace) / rmsu`
	`powerfac_2 = cos(phaseShift_2) ; the same as cosPhi`
	
	`; 3.) some more information`
	`; cosPhi = RealPower / ApparentPower`
	`; cosPhi = PowerFactor`
	`; cosPhi = R / Z  = resistance / impedance = Wirkwiderstand / Scheinwiderstand`

CrestFactorPeriodic(a [, start, end])	Berechnet den Scheitelfaktor (Crest-Faktor) ganzer Perioden von **a**. Dieser beschreibt das Verhältnis vom absoluten Spitzenwert zu periodischen RMS des Parameters **a**. Der Rückgabewert ist ein Skalar vom Typ Double. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll.
	
	Example:
	`; Generate two test traces, phase shift of current`
	`deltaPhase = 20 ; would be 20° which indicates an inductive load`
	`voltageTrace = Sinus(330, 50, 0, 1e6, 1e6)`
	`currentTrace = Sinus(10, 50, -1*deltaPhase, 1e6, 1e6)`
	
	`; 1.) Formula Editor power functions`
	`cosphi_1 = CosPhi(voltageTrace, currentTrace)`
	`ApparentPower_1[VA] = ApparentPower(voltageTrace, currentTrace)`
	`RealPower_1[W] =  RealPower(voltageTrace, currentTrace)`
	`ReactivePower_1[VAR] = ReactivePower(voltageTrace, current-Trace)`

	`crestfac_1 = CrestFactorPeriodic(voltageTrace)`
	`powerfac_1 = PowerFactor(voltageTrace, currentTrace)`
	
	`; 2.) Manual calculation`
	`phaseTrace = Phase(voltageTrace, currentTrace)`
	`phaseShift_2[°] = phaseTrace(TEnd(phaseTrace))`
	`rmsU[V] = RMSP(voltageTrace)`
	`rmsI[A] = RMSP(currentTrace)`
	
	`cosphi_2 = cos(phaseShift_2)`
	`ApparentPower_2[VA] = rmsU* rmsI`
	`RealPower_2[W] =  rmsU* rmsI * cos(phaseShift_2)`
	`ReactivePower_2[VAR] = rmsU* rmsI * sin(phaseShift_2)`
	`crestfac_2 = max(voltageTrace) / rmsu`
	`powerfac_2 = cos(phaseShift_2) ; the same as cosPhi`
	
	`; 3.) some more information`
	`; cosPhi = RealPower / ApparentPower`
	`; cosPhi = PowerFactor`
	`; cosPhi = R / Z  = resistance / impedance = Wirkwiderstand / Scheinwiderstand`

FundamentalPower(a, b [, start, end [, level, hysteresis]])	Berechnet die Wirkleistung der Grundwelle, ohne den Anteil der Oberwellen aus den beiden Kurven **a** und **b**. Der Rückgabewert ist ein Skalar vom Typ double. Um sinnvolle Resultate zu erhalten, sollten die Signale periodisch sein und die eine Kurve eine Spannung und die andre ein Strom darstellen. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll. Weitere optionale Parameter sind **level** und **hysteresis**. Sind diese nicht definiert, wird **level** auf den Mittelwert des Spitzenwerts gelegt, **hysteresis** beträgt 5% des Spitzenwerts.
	
	Example:
	`tr1 = c0a1`
	`tr2 = c0a2`
	`fPower = FundamentalPower(tr1,tr2)`

PowerFactor(a, b [, start, end [, level, hysteresis]])	Berechnet das Verhältnis von Wirkleistung zu Scheinleistung (Real- zu Apparent-Power) von **a** und **b**. Ein Parameter sollte die Spannung, der andere der Strom sein. Der Rückgabewert ist ein Skalar vom Typ double. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll. Weitere optionale Parameter sind **level** und **hysteresis**. Sind diese nicht definiert, wird **level** auf den Mittelwert des Spitzenwerts gelegt, **hysteresis** ist bei 5% des Spitzenwerts.
	
	Example:
	`; Generate two test traces, phase shift of current`
	`deltaPhase = 20 ; would be 20° which indicates an inductive load`
	`voltageTrace = Sinus(330, 50, 0, 1e6, 1e6)`
	`currentTrace = Sinus(10, 50, -1*deltaPhase, 1e6, 1e6)`
	
	`; 1.) Formula Editor power functions`
	`cosphi_1 = CosPhi(voltageTrace, currentTrace)`
	`ApparentPower_1[VA] = ApparentPower(voltageTrace, currentTrace)`
	`RealPower_1[W] =  RealPower(voltageTrace, currentTrace)`
	`ReactivePower_1[VAR] = ReactivePower(voltageTrace, current-Trace)`
	`crestfac_1 = CrestFactorPeriodic(voltageTrace)`

	`powerfac_1 = PowerFactor(voltageTrace, currentTrace)`
	
	`; 2.) Manual calculation`
	`phaseTrace = Phase(voltageTrace, currentTrace)`
	`phaseShift_2[°] = phaseTrace(TEnd(phaseTrace))`
	`rmsU[V] = RMSP(voltageTrace)`
	`rmsI[A] = RMSP(currentTrace)`
	
	`cosphi_2 = cos(phaseShift_2)`
	`ApparentPower_2[VA] = rmsU* rmsI`
	`RealPower_2[W] =  rmsU* rmsI * cos(phaseShift_2)`
	`ReactivePower_2[VAR] = rmsU* rmsI * sin(phaseShift_2)`
	`crestfac_2 = max(voltageTrace) / rmsu`
	`powerfac_2 = cos(phaseShift_2) ; the same as cosPhi`
	
	`; 3.) some more information`
	`; cosPhi = RealPower / ApparentPower`
	`; cosPhi = PowerFactor`
	`; cosPhi = R / Z  = resistance / impedance = Wirkwiderstand / Scheinwiderstand`

ReactivePower(a, b [, start, end [, level, hysteresis]])	Berechnet die Blindleistung aus **a** und **b**, beide sind Kurven, von denen die eine die Spannung und die andere der Strom darstellt. Der Rückgabewert ist ein Skalar vom Typ double. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll. Weitere optionale Parameter sind **level** und **hysteresis**. Sind diese nicht definiert, wird **level** auf den Mittelwert des Spitzenwerts gelegt, **hysteresis** ist bei 5% des Spitzenwerts.
	
	Example:
	`; Generate two test traces, phase shift of current`
	`deltaPhase = 20 ; would be 20° which indicates an inductive load`
	`voltageTrace = Sinus(330, 50, 0, 1e6, 1e6)`
	`currentTrace = Sinus(10, 50, -1*deltaPhase, 1e6, 1e6)`
	
	`; 1.) Formula Editor power functions`
	`cosphi_1 = CosPhi(voltageTrace, currentTrace)`
	`ApparentPower_1[VA] = ApparentPower(voltageTrace, currentTrace)`
	`RealPower_1[W] =  RealPower(voltageTrace, currentTrace)`
	`ReactivePower_1[VAR] = ReactivePower(voltageTrace, current-Trace)`
	`crestfac_1 = CrestFactorPeriodic(voltageTrace)`
	`powerfac_1 = PowerFactor(voltageTrace, currentTrace)`
	
	`; 2.) Manual calculation`
	`phaseTrace = Phase(voltageTrace, currentTrace)`
	`phaseShift_2[°] = phaseTrace(TEnd(phaseTrace))`
	`rmsU[V] = RMSP(voltageTrace)`
	`rmsI[A] = RMSP(currentTrace)`
	
	`cosphi_2 = cos(phaseShift_2)`
	`ApparentPower_2[VA] = rmsU* rmsI`
	`RealPower_2[W] =  rmsU* rmsI * cos(phaseShift_2)`
	`ReactivePower_2[VAR] = rmsU* rmsI * sin(phaseShift_2)`
	`crestfac_2 = max(voltageTrace) / rmsu`
	`powerfac_2 = cos(phaseShift_2) ; the same as cosPhi`
	
	`; 3.) some more information`
	`; cosPhi = RealPower / ApparentPower`
	`; cosPhi = PowerFactor`

	`; cosPhi = R / Z  = resistance / impedance = Wirkwiderstand / Scheinwiderstand`

RealPower(a, b [, start, end [, level, hysteresis]])	Berechnet die Wirkleistung vom Mittelwert der Momentan-Leistung von **a** und **b**. Einer der Parmeter sollte die Spannung und der andere Strom sein. Der Rückgabewert ist ein Skalar vom Typ double. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll. Weitere optionale Parameter sind **level** und **hysteresis**. Sind diese nicht definiert, wird **level** auf den Mittelwert des Spitzenwerts gelegt, **hysteresis** ist bei 5% des Spitzenwerts.
	
	Example:
	`; Generate two test traces, phase shift of current`
	`deltaPhase = 20 ; would be 20° which indicates an inductive load`
	`voltageTrace = Sinus(330, 50, 0, 1e6, 1e6)`
	`currentTrace = Sinus(10, 50, -1*deltaPhase, 1e6, 1e6)`
	
	`; 1.) Formula Editor power functions`
	`cosphi_1 = CosPhi(voltageTrace, currentTrace)`
	`ApparentPower_1[VA] = ApparentPower(voltageTrace, currentTrace)`
	`RealPower_1[W] =  RealPower(voltageTrace, currentTrace)`
	`ReactivePower_1[VAR] = ReactivePower(voltageTrace, current-Trace)`
	`crestfac_1 = CrestFactorPeriodic(voltageTrace)`
	`powerfac_1 = PowerFactor(voltageTrace, currentTrace)`
	
	`; 2.) Manual calculation`
	`phaseTrace = Phase(voltageTrace, currentTrace)`
	`phaseShift_2[°] = phaseTrace(TEnd(phaseTrace))`
	`rmsU[V] = RMSP(voltageTrace)`
	`rmsI[A] = RMSP(currentTrace)`
	
	`cosphi_2 = cos(phaseShift_2)`
	`ApparentPower_2[VA] = rmsU* rmsI`
	`RealPower_2[W] =  rmsU* rmsI * cos(phaseShift_2)`
	`ReactivePower_2[VAR] = rmsU* rmsI * sin(phaseShift_2)`
	`crestfac_2 = max(voltageTrace) / rmsu`
	`powerfac_2 = cos(phaseShift_2) ; the same as cosPhi`
	
	`; 3.) some more information`
	`; cosPhi = RealPower / ApparentPower`
	`; cosPhi = PowerFactor`
	`; cosPhi = R / Z  = resistance / impedance = Wirkwiderstand / Scheinwiderstand`

TotalHarmonicDistortion(a [, start, end [, level, hysteresis]])	Sucht die Grundfrequenz in **a** und berechnet daraus den Wert THD (Total Harmonic Distortion) in % eines periodischen Signals. Der Rückgabewert ist ein Skalar vom Typ double in Prozent. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll. Weitere optionale Parameter sind **level** und **hysteresis**. Sind diese nicht definiert, wird **level** auf den Mittelwert des Spitzenwerts gelegt, **hysteresis** ist bei 5% des Spitzenwerts. Um sinnvolle Resultate zu erreichen sollten mindestens zwei Perioden vorhanden sein. **Example:**

	`trace = c0A1` `THD = TotalHarmonicDistortion(trace)`

34.14 Programming Functions

Calculate(filepath, waitForCalculation, showResults)	Berechnet eine Formel *.for innerhalb einer Formel. Der Rückgabewert **variable** ist vom Typ Dictionary und beinhaltet die berechneten Werte der aufgerufenen Formel.
	filepath ist die zu berechnende Formeldatei mit der Endung *.for. Es können auch Formeln aus ExperimentSets *.zip ausgeführt werden. Der Parameter **waitForCalculation** definiert ob auf die Formel gewartet wird. **waitForCalculations** auf **False** gesetzt kann dazu verwendet werden, wenn die berechneten Werte nicht innerhalb dieser Formel verwendet werden, z.B. Update im Waveform oder andere losgelöste Funktionen.
	showResults zeigt die berechneten Werte in Traces und Results Tab.
	
	Example:
	`path = GetPath("experimentset")`
	`path = PathCombine(path, "Formula 2.for")`
	
	`; contains dictionary of results from Formula 2.for`
	`retval = Calculate(path, true, true)`
	`globalVar = ReadOfPrevious("GLOBALVAR")`
	
	`pause(globalVar)`
	
	`EndFormula`
	
	`; Create a second Formula with the name Formula 2.for`
	`; Copy and paste the code below into this file`
	`; Save file with Ctrl+s`
	`a = 5`
	`tr = Sinus(10,50,0,10E3,1E3)`
	
	`mylist = List()`
	`mylist() = 12`
	`mylist() = "Example"`
	
	`WriteForNext("GLOBALVAR","Formula 2 calculated!")`

Delay(seconds)	Verzögert den Ablauf entsprechend dem Parameter **seconds**.
	
	Diese Funktion liefert keine genaue zeitliche Verzögerung und sollte deshalb nicht für genaue Zeitmessungen eingesetzt werden. Die Abweichungen hängen von parallellaufenden Prozessen im Betriebssystem ab. Diese können einige ms betragen.
	Example:
	`Delay(1)  ; 1 second`
	`Delay(100E-3) ; 100 miliseconds`

Do-Until	Generiert eine Schlaufe welche **block** so lange ausführt bis die Bedingung **a=0** richtig ist. Die Bedingung kann die folgenden Operatoren verwenden: "=", ">", "<", ">=", "<=", "<>". Mehrere Bedingungen können mit den Wörtern **and**, **or** und **not** zusammengehängt werden.

	Der Operator **<>** entspricht dem Ausdruck **not equal** (!=)
	Example:
	`i = 100`
	`do until i = 15`
	`  i = i-1`
	`loop`

Do-While	Erstellt eine Schlaufe welche **block** mindestens einmal ausführt. Die Bedingung **a=0** wird am Schluss der Schlaufe ausgewertet und entscheidet ob diese beendet wird. Die Bedingung kann die folgenden Operatoren verwenden: "=", ">", "<", ">=", "<=", "<>". Mehrere Bedingungen können mit den Wörtern **and**, **or** und **not** zusammengehängt werden.
	Der Operator **<>** entspricht dem Ausdruck **not equal** (!=)
	Example:
	`i = 0`
	`do`
	`  i = i + 1`
	`loop while i < 100`

EndFormula	Diese Operation dient als Abschluss der Berechnungen. Wenn sie fehlt werden alle Formel-Linien bis ans Ende abgearbeitet. Sie ist vor allem nützlich bei der Entwicklung von Formel-Programmen. Man kann damit verhindern, dass nachfolgende (ev. noch fehlerhafte) Formeln gelöscht oder auskommentiert werden müssen.
	Example:
	`sineTrace = Sinus(1, 1e3, 0, 1e6, 1e4)`
	`; We stop the calculation from this line`
	`EndFormula`
	`; Add an offset to the curve`
	`sineTrace = sineTrace + 10`

Exitloop	Mit dieser Operation kann eine **Loop-Schlaufe** verlassen werden.
	Example:
	`for i=1 to 4 step 1`
	`  if i=3 then`
	`  exitloop`
	`  endif`
	`next`

False	Schlüsselwort **False**, entspricht dem Wert 0. Es signalisiert, dass eine Bedingung nicht erfüllt ist.
	Example:
	`if FileExist("xy.tpc5") <> False then`
	`  BLOCK`
	`endif`

For-Each

Mit der Schlaufe **for each** können sehr einfach zählbare Elemente durchgegangen werden. Es werden alle einzelnen Elemente **item** von der Enumeration **enumerable** zurückgegeben. Folgende Datentypen können verwendet werden: List, Dictionary, Array, String, Trace.

> Wenn **Dictionary** für die Enumeration verwendet wird enthält **item** den Schlüssel jedes Wertepaares.

Example:

```
Pause("Start Multi Block recording with 5 blocks. Click continue when done.")

; 1. Example
; get maximum signal from each block
maxVal = NotDefined
for each item in EnBlocks(c0A1,false)
    Mx = Max(item)

    if (Mx > maxVal) or (maxVal = NotDefined) then
      maxVal = Mx
    endif
next


; 2. Example
; - Analyse hardware channel
; - Data from reference
; - Data from saved file

; Save channel A1 to a file
Save("myMultiBlockSignal.tpc5", "0A1") ; saves all recorded blocks
; Please note: This saves just the first block!
Save("mytest.tpc5", c0A1)
; Set channel A1 to Reference 1
SetReference(ref1,true, c0A1) ; true means references to the hardware channel

n1 = NBlks(c0A1)
n2 = NBlks(ref1)
n3 = NBlks(File("myMultiBlockSignal.tpc5",0))

cnt1 = 0
cnt2 = 0
cnt3 = 0

minVal1 = List()
minVal2 = List()
minVal3 = List()

; Hardware channel
for each item in EnBlocks(c0A1,false)
  cnt1 += 1
  tr1 = item
  minVal1() = Min(item)
next

; Reference
for each item in EnBlocks(ref1,false)
  cnt2 += 1
  tr2 = item
  minVal2() = Min(item)
next

; From File
```

```
for each item in EnBlocks(File("myMultiBlockSig-
nal.tpc5",0),false)
  cnt3 += 1
  tr3 = item
  minVal3() = Min(item)
next
```

For-Loop

Erstellt eine Schlaufe welche **block** ausführt so lange die Bedingung **a <= limit** oder **a >= limit** richtig ist. Die Variabel **a** wird am Schluss der Schlaufe inkrementiert mit dem Schrittwert **b**.

> Es muss gewährleistet werden, dass die Schlaufe nicht in eine Endlosschlaufe endet falls die Variabel **a** innerhalb der Schlaufe verändert wird.

Example:

```
Pause("Start Multi Block recording with 5 blocks. Click conti-
nue when done.")

; 1. Example
; get maximum signal from each block
maxVal = NotDefined
for i = 0 to NBlks(c0A1)-1
    Mx = Max(c0A1.i)

    if (Mx > maxVal) or (maxVal = NotDefined) then
      maxVal = Mx
    endif
next


; 2. Example
; - Analyse hardware channel
; - Data from reference
; - Data from saved file

; Save channel A1 to a file
Save("myMultiBlockSignal.tpc5", "0A1") ; saves all recorded
blocks
; Please note: This saves just the first block!
Save("mytest.tpc5", c0A1)
; Set channel A1 to Reference 1
SetReference(ref1,true, c0A1) ; true means references to the
hardware channel

n1 = NBlks(c0A1)
n2 = NBlks(ref1)
n3 = NBlks(File("myMultiBlockSignal.tpc5",0))

cnt1 = 0
cnt2 = 0
cnt3 = 0

minVal1 = List()
minVal2 = List()
minVal3 = List()

; Hardware channel
for a = 0 to n1-1 step 1.0
  tr1 = c0A1.a
  minVal1() = Min(tr1)
next

; Reference
for a = 0 to n1-1 step 1.0
  tr2 = ref1.a
  minVal2() = Min(tr2)
next
```



```
; From File
for a = 0 to n1-1 step 1.0
  tr3 = File("myMultiBlockSignal.tpc5",0).a
  minVal3() = Min(tr3)
next
```

Function

Function ist ein Schlüsselwort, welches den Anfang eines Formel-Block **block** definiert wird. Nach dem Schlüsselwort wird der Name **name** der Funktion erwartet. Zwischen den geschweiften Klammern werden alle Parameter definiert. Die Anzahl der Parameter ist nicht limitiert. Es sind alle Variablen-Typen (Zahl, Kurve, Array, Liste usw.) als Parameter zulässig. **EndFunction** schliesst die Funktion (**block**) ab.

> **Achtung:** Die Parameter werden als Referenz übergeben. Sie können also in einer Funktion direkt neu berechnet und zurückgegeben werden. Wenn dies geschieht, **gilt zu beachten**, dass diese Variablen auch im **Haupt-Programm verändert** werden.

Der direkte Rückgabewert muss dem Namen **name** der Funktion zugewiesen werden. Als Rückgabewert sind alle Variablen-Typen (Zahl, Kurve, Array, Liste usw.) zulässig.
Siehe auch die Funktionen **Using** und **Endformula**.

Example:

```
; Example: Function inside the same Formula
x = 5
y = 7
z = Addition(x,y)

Endformula

Function Addition(value0, value1)
  Addition = value0 + value1
EndFunction
```

GetEnvironment(key)

Liest anhand **key** Umgebungsvariabeln aus der aktuellen Formel. Der Rückgabewert **variable** ist ebenfalls vom Typ Zahl.
Parameter **key** hat die folgenden Funktionen:
0 = UseMemory (Siehe auch die Funktion **UseMemory**)
1 = AlwaysCloseFiles
5 = DontSaveHdf
6 = IgnoreDataAvailability
7 = UpdateGUI
8 = IsDebug
9 = IgnoreAbortDuringTryCatch
Parameter 1, 6 und 7 sind per Default aktiv, alle anderen inaktiv.

Example:

```
; disable writing files to expr folder
SetEnvironment(0, True) ; switch ON
tr1 = Sinus(10,50,0,1E3,1E3) ; Tr1 is saved in RAM
SetEnvironment(0, False) ; switch OFF
tr2 = tr1 ; Tr2 is saved as TPC5 file

; read changes of function UseMemory()
```

<table>
<tr><td></td><td>

```
val1 = GetEnvironment(0)
UseMemory(True)
val2 = GetEnvironment(0)
UseMemory(False)
val3 = GetEnvironment(0)
```

</td></tr>
<tr><td>

GetVariableValue(name [, useGlobalScope])

</td><td>

Liefert den Wert der Variablen mit dem Namen **name**.
Falls die Variable nicht vorhanden ist, wird **NotDefined** zurückgegeben.
Der optionale Parameter **useGlobalScope** kann verwendet werden um innerhalb eines Funktionsaufrufes auf den globalen Scope zuzugreifen.

> Diese Funktion wird meistens nach der Funktion **SetVariableValue** benötigt.

Example:

```
; create some variables
for a = 0 to 10 step 1.0
   str = "myvariable_"+a
   SetVariableValue(str, a^2)
next

; read back the variables
mylist = List()
for a = 0 to 12 step 1.0
   str = "myvariable_"+a
   val = GetVariableValue(str)
   if not (val = NotDefined) then ; check if valid or not
       mylist() = val
    endif
next
```

</td></tr>
<tr><td>

If-Else

</td><td>

Erstellt einen Entscheidungszweig welcher dafür verwendet werden kann um verschiedene Fälle (**block1**, **block2** und **block3**) in einer Formel abzudecken.
Wenn die Bedingung **i = b** erfüllt ist wird die Formel das if-Statement ausführen, ansonsten das else-Statement. Die Bedingungen können folgende Operatoren verwenden:
"=", ">", "<", ">=", "<=", "<>".
Mehrere Bedingungne können mit den Wörtern **and**, **or** und **not** verknüpft werden.

> Der Operator **<>** entspricht **not equal** (!=)

Wenn man weitere Entscheidungszweige benötigt kann man weitere "**elseif i < b then**" nach dem if-Statement verwenden. Es gibt keine Begrenzung wieviele **elseif** Statements geschrieben werden können.

Example:

```
if Mx > Bigest then
    Bigest=Mx
else
    Bigest = 0
endif
```

</td></tr>
</table>

IsDebug()

Prüft ob eine Formel im Debug Modus ausgeführt wird. Der Rückgabewert **variable** ist vom Typ Boolean. **Debug Modus aktiv= True**, Normaler Modus = False.

GetEnviroment(8) gibt den selben Wert zurück.

Example:

```
if IsDebug()=1 then
    ; do some debug stuff
endif
```

lock-endlock

Sperrt andere Prozesse bei parallelen Berechnungen.
Diese Funktion wird in Kombination mit **parallelen Rechenaufgaben** wie beim **parfor** Statement verwendet. Der Code **block** zwischen lock und endlock wird dann gesperrt von den anderen parallelen Prozessen ausgeführt, so dass diese **anderen Prozesse keinen Einfluss** auf z.B. ein Zwischenresultat haben.
Siehe auch die Funktion **parfor** und **parfor-each**.

Example:

```
; Scope recording, maximun block length for this example

UseMemory(True)
tr = c0A1

maxValParFor = NotDefined
tasks = 4 ; number of tasks
lengthMeas = Length(tr) ; number of samples
dt = TSample(tr) ; time between samples
subCount = (lengthMeas / tasks)*dt ; duration of signal for one
task
t1 = TBegin(tr) ; time at beginning of trace

StartWatch()
parfor index = 0 to (tasks-1)
    tStart = t1 + (index * subCount)
    tStop = tStart + (subCount -dt)
    maxVal = Max(tr, tStart, tStop)

    ; In order to access the variable "maxValParFor" of several
tasks, this must take place between a lock statement.
    ; Otherwise, it can happen that the tasks overwrite each
other and thus the correct maximum value in the variable is not
available.
    lock
        if maxValParFor = NotDefined then
           maxValParFor = maxVal
        elseif maxValParFor < maxVal then
            maxValParFor = maxVal
        endif
    endlock
next
tParFor[s] = StopWatch()/1000
SetUnit(maxValParFor, GetUnit(tr))


; Single task operation
StartWatch()
maxValStd = Max(tr)
tStandard[s] = StopWatch()/1000
SetUnit(maxValStd, GetUnit(tr))
```

	`pause("Single task vs. four tasks", lengthMeas, tStandard, tParFor, maxValStd, maxValParFor)`

NotDefined	Schlüsselwort, Rückgabewert von Funktionen, welche keinen gültigen Wert ermitteln konnten.
	
	Example:
	`tx=TCross(trc,-10,10,0.5)`
	
	`If tx = NotDefined Then`
	` ; No crossing found`
	`Endif`

parfor-each	Mit der **parfor each** Funktionen, können Berechnungen auf mehrere Kerne einer CPU aufgeteilt werden. Die verkürzt die Rechenzeit und nutzt das mögliche Potential von Mehrkernprozessoren besser aus.
	Die beiden Funktionen **Lock** und **EndLock** müssen zur Synchronisation gemeinsamer Ressourcen, wie im Beispiel ein Dictionary, verwendet werden.
	
	Siehe auch die Funktion **parfor** und **lock-endlock**.
	
	Geschwindigkeitsvorteile werden nur erziehlt, wenn eine grosse Sample-Anzahl pro Kanal verarbeitet wird. Parallele-Schlaufen sollten im generellen nur verwendet werden, wenn eine grosse Arbeitslast vorliegt. Ansonsten kann ein Nachteil entstehen in der Geschwindigkeit, da dass wechseln der Task einen Overhead erzeugt.
	
	Hier sind generelle Regeln die verwendet werden können um zu bestimmen, ob eine parallele Schlaufe sich lohnt:
	
	• Grosse Arbeitslast sollten vorhanden sein (viele Dateien, grosse Anzahl-Samples, komplexe Algorithmen, etc.)
	• Lesen von Daten direkt vom PC ist viel schneller als das Lesen vom Netzwerk. Direktes lesen sollte wenn immer bevorzugt werden.
	• Speichern von Daten in ein *.tpc5 wenn möglich um dieses danach mit höchstmöglichem Durchsatz zu analysieren.
	
	Example:
	`UseMemory(True)`
	`values = Dictionary()`
	
	`parfor each input in enChannels()`
	`  desc = GetChannelDesc(input)`
	`  key = "" + desc(0) + ("A"C + desc(1)) + desc(2)`
	`  tMax = TOfMax(input)`
	`  tMin = TOfMin(input)`
	`  traceSlice = Slice(input, tMax, tMin)`
	`  lowPassTrace = LowPass(traceSlice, Bessel, 2, 1e3)`
	`  meanVal = Mean(lowPassTrace)`
	
	`  Lock`
	`  values(key) = meanVal`
	`  EndLock`

	`next`
parfor	Vergleichbar mit der **for next** Funktion. **Parfor next** kann auf Mehrkernprozessoren Systemen verwendet werden, um die Geschwindigkeit für Berechnungen dank paralleler Abarbeitung zu steigern. **Parfor next** kann zum Beispiel dazu verwendet werden, parallel mehrere gespeicherte Dateien zu analysieren und deren Werte in ein Dictionary zu speichern. Siehe auch die Funktion **parfor-each** und **lock-endlock**. Geschwindigkeitsvorteile werden nur erziehlt, wenn eine grosse Sample-Anzahl pro Kanal verarbeitet wird. Parallele-Schlaufen sollten im generellen nur verwendet werden, wenn eine grosse Arbeitslast vorliegt. Ansonsten kann ein Nachteil entstehen in der Geschwindigkeit, da dass wechseln der Task einen Overhead erzeugt. Hier sind generelle Regeln die verwendet werden können um zu bestimmen, ob eine parallele Schlaufe sich lohnt: • Grosse Arbeitslast sollten vorhanden sein (viele Dateien, grosse Anzahl-Samples, komplexe Algorithmen, etc.) • Lesen von Daten direkt vom PC ist viel schneller als das Lesen vom Netzwerk. Direktes lesen sollte wenn immer bevorzugt werden. • Speichern von Daten in ein *.tpc5 wenn möglich um dieses danach mit höchstmöglichem Durchsatz zu analysieren. **Example:**

```
; Scope recording, maximun block length for this example

UseMemory(True)
tr = c0A1

maxValParFor = NotDefined
tasks = 4 ; number of tasks
lengthMeas = Length(tr) ; number of samples
dt = TSample(tr) ; time between samples
subCount = (lengthMeas / tasks)*dt ; duration of signal for one
task
t1 = TBegin(tr) ; time at beginning of trace

StartWatch()
parfor index = 0 to (tasks-1)
    tStart = t1 + (index * subCount)
    tStop = tStart + (subCount -dt)
    maxVal = Max(tr, tStart, tStop)

    ; In order to access the variable "maxValParFor" of several
tasks, this must take place between a lock statement.
    ; Otherwise, it can happen that the tasks overwrite each
other and thus the correct maximum value in the variable is not
available.
    lock
        if maxValParFor = NotDefined then
```

<table>
<tr><td></td><td>

```
            maxValParFor = maxVal
        elseif maxValParFor < maxVal then
            maxValParFor = maxVal
        endif
    endlock
next
tParFor[s] = StopWatch()/1000
SetUnit(maxValParFor, GetUnit(tr))


; Single task operation
StartWatch()
maxValStd = Max(tr)
tStandard[s] = StopWatch()/1000
SetUnit(maxValStd, GetUnit(tr))

pause("Single task vs. four tasks", lengthMeas, tStandard,
tParFor, maxValStd, maxValParFor)
```

</td></tr>
<tr><td>

ReadOfPrevious(key)

</td><td>

ReadOfPrevious dient zusammen mit WriteForNext als seine Art globale Variable. **key** ist ein String und der Name der Variable.

Auf die in der Variablen gespeicherten Werte kann bei jedem erneuten Durchlauf der Formeln zugegriffen werden. So können Berechnungen über mehrere Messungen hinweg durchgeführt werden.

Example:

```
; Averaging
; get last trace
mytrace = ReadValueOfPrevious("mytrace")

; measure and smooth new trace
tr = c0A1;
tr = Smooth(tr, 100)

; first run?
if mytrace = NotDefined then
  ; yes, first run
  mytrace = tr
else
  ; no, not the first run
  mytrace = (mytrace + tr) /2
endif

; store value for next calculation
WriteValueForNext("mytrace",mytrace)
```

</td></tr>
<tr><td>

SetEnvironment(key, value)

</td><td>

Setzt spezielle Parameter zur Optimierung von TranAX Performance aus.

Parameter **key** hat die folgenden Funktionen:

- 0 = UseMemory (Siehe auch die Funktion **UseMemory**) - **value** => Bool
- 1 = AlwaysCloseFiles - **value** => Bool
- 5 = DontSaveHdf - **value** => Bool
- 6 = IgnoreDataAvailability - **value** => Bool
- 7 = UpdateGUI - **value** => Bool
- 8 = IsDebug - **value** => Bool
- 9 = IgnoreAbortDuringTryCatch - **value** => Bool

</td></tr>
</table>

	Parameter 1, 6 und 7 sind per Default aktiv, alle anderen inaktiv.
	
	Example:
	`; disable writing files to expr folder`
	`SetEnvironment(0, True) ; switch ON`
	`tr1 = Sinus(10,50,0,1E3,1E3) ; Tr1 is saved in RAM`
	`SetEnvironment(0, False) ; switch OFF`
	`tr2 = tr1 ; Tr2 is saved as TPC5 file`
	
	`; read changes of function UseMemory()`
	`val1 = GetEnvironment(0)`
	`UseMemory(True)`
	`val2 = GetEnvironment(0)`
	`UseMemory(False)`
	`val3 = GetEnvironment(0)`

SetMessage(text [, useOverlayWindow])	Schreibt die Nachricht **text** vom Typ String in die Statusanzeige des Formel Editors unten rechts in TranAX. Der optionale Parameter **useOverlayWindow** vom Typ Boolean bietet die Möglichkeit, die Nachricht entweder in der Statusanzeige oder in der Mitte des TranAX Fensters anzuzeigen. Ist **useOverlayWindow** gleich True, wir die Meldung Angezeigt und blendet langsam wieder aus. Ist dieser gleich False wird die Nachricht, wie ohne optionale Parameter, unten rechts angezeigt.
	
	Example:
	`; message in status bar`
	`SetMessage("my message")`
	
	`; message as window`
	`SetMessage("my message", true)`

SetVariableValue(name, value [, useGlobalScope])	Setzt oder erstellt eine Variable mit dem Namen**name**. Falls bereits eine gleichnamige Variable existiert wird diese auf den neuen Wert gesetzt. Der Parameter **value** wird der Variabel mit dem Namen **name** zugewiesen. Wird der optionale Parameter **useGlobalScope** auf **True** gesetzt, schreibt die Formel den Wert **value** von der Variabel **name** in den globalen Scope.
	
	Um den Wert einer gesetzten oder defnierten Variablen zu erhalten, kann die Funktion **GetVariableValue** verwendet werden.
	
	Example:
	`; create some variables`
	`for a = 0 to 10 step 1.0`
	`  str = "myvariable_"+a`
	`  SetVariableValue(str, a^2)`
	`next`
	
	`; read back the variables`
	`mylist = List()`
	`for a = 0 to 12 step 1.0`
	`  str = "myvariable_"+a`
	`  val = GetVariableValue(str)`
	`  if not (val = NotDefined) then ; check if valid or not`
	`  mylist() = val`
	`  endif`
	`next`

SplitString(text, symbol)	Trennt den String **text** mit dem Zeichen **symbol** und liefert ein Array mit den jeweiligen Teilstrings (das Zeichen **symbol** existiert nicht mehr). **Symbol** kann auch aus mehreren Zeichen bestehen. Der ursprüngliche String **text** wird dann an der entsprechenden Stelle getrennt. Falls **symbol** mehrfach vorkommt, in **text** wird dieser in mehrere Teil-Strings gewandelt. Das Array ist dann entsprechend länger. **Example:** `SplitString(text, symbol)`

StopCalculation(filepath)	Stoppt die Berechnung einer Formel, welche mit der Funktion **Calculate** gestartet wurde. **Filepath** ist die zu berechnende Formeldatei mit der Endung *.for. Es können auch Formeln aus ExperimentSets *.zip ausgeführt werden. Rückgabewert ist ein Integer. Wenn die Formel gestoppt wurde, ist der Rückgabewert = 1. Falls die Formel nicht am Berechnen war ist der Rückgabe wert = 0.
	
	Example:
	`; contains dictionary of results from Formula 2.for`
	
	`myFormula = "Empty 123.zip\Formula 2.for"`
	`Calculate(myFormula, false, true)`
	`Delay(5)`
	`val = StopCalculation(myFormula)`
	
	`EndFormula`
	
	`; Create a second Formula with the name Formula 2.for`
	`; Copy and paste the code below into this file`
	`; Save file with Ctrl+s`
	`cntr = 0`
	
	`do while cntr < 1000`
	`  cntr += 1`
	`  Delay(0.1)`
	`loop`

StringFormat(a, format)	Konvertiert **a** in einen String. Dessen Formatierung **format** kann als String definiert werden.
	
	Die Beispiele zeigen ausführlich wie man die Funktion für Datum, Zeitdifferenzen, Zahlen, etc. verwendet
	
	Example:
	`; Date Time Formats`
	`; ----------`
	`dateTimeNow = GetDateTime()`
	`dateTime_d = StringFormat(dateTimeNow, "d") ; d -> Short Date -> 10/04/2019`
	`dateTime_DU =StringFormat(dateTimeNow, "D") ; D -> Long Date -> Friday, 04 October 2019`
	`dateTime_t =StringFormat(dateTimeNow, "t")  ; t -> Short Time -> 14:20`
	`dateTime_TU =StringFormat(dateTimeNow, "T") ; T -> Long Time -> 14:20:44`
	`dateTime_f =StringFormat(dateTimeNow, "f")  ; f -> Long Date Time -> Friday, 04 October 2019 14:20`
	`dateTime_FU =StringFormat(dateTimeNow, "F") ; F -> Long Date Time (seconds) -> Friday, 04 October 2019 14:20:44`

	`dateTime_g =StringFormat(dateTimeNow, "g") ; g -> Short Date Time -> 10/04/2019 14:20`
	`dateTime_GU =StringFormat(dateTimeNow, "G") ; G -> Short Date Time (seconds) 10/04/2019 14:20:44`
	`dateTime_M =StringFormat(dateTimeNow, "M") ; M -> Short Date -> October 04`
	`dateTime_r =StringFormat(dateTimeNow, "r") ; r -> RFC1123 Date Time String -> Fri, 04 Oct 2019 14:20:44 GMT`
	`dateTime_s =StringFormat(dateTimeNow, "s") ; s -> Sortable Date Time String -> 2019-10-04T14:20:44`
	`dateTime_u =StringFormat(dateTimeNow, "u") ; u -> Universal Sortable Date -> 2019-10-04 14:20:44Z`
	`dateTime_UU =StringFormat(dateTimeNow, "U") ; U -> Universal full date -> Friday, 04 October 2019 04:20:44`
	`dateTime_Y =StringFormat(dateTimeNow, "Y") ; Y -> Year month pattern -> 2019 October`
	`dateTime_custom = StringFormat(dateTimeNow,"dd.MM.yyyy HH:mm:ss") ; dd: Days, MM: Month, yyyy: Year, HH: Hours, mm: Minutes, ss: Seconds`
	
	`; Number Formats`
	`; ----------`
	`number = 7656.456`
	`number_c = StringFormat(number,"c") ; c -> currency -> $7,656.46`
	`number_e = StringFormat(number,"e3") ; e -> Scientific (number = digits after decimal point) -> 7.656e+003`
	`number_f = StringFormat(number,"f2") ; f -> Fixed Point (number = digits after decimal point) -> 7656.46`
	`number_g = StringFormat(number,"g5") ; g -> General (number = total amount of digits) -> 7656.5`
	`number_n = StringFormat(number,"n4") ; n -> Thousand Separator -> 7,656.4560`
	
	`; Custom Formats`
	`; ----------`
	`number = 81456.3467`
	`number_0 = StringFormat(number, "00.00") ; 0 -> Zero Placeholder -> 81456.35`
	`number_hash = StringFormat(number, "#.#####") ; 0 -> Digit Placeholder -> 81456.3467`
	`number_point = StringFormat(number, "0.0") ; 0 -> Decimal Point -> 81456.3`
	`number_sep = StringFormat(number, "0,0") ; 0 -> Thousand Separator -> 81,456`
	`number_per = StringFormat(0.758, "0%") ; 0 -> Percent -> 76%`

StringToNumber(text)	Wandelt den String **text** in eine Zahl. Vorausgesetzt ist natürlich, dass es sich um eine Zahl im String handelt. Ansonsten wird **NotDefined** zurück geliefert. **Example:** `StringToNumber(text)`

True	Schlüsselwort **True**, entspricht dem Wert 1. Es signalisiert, dass eine Bedingung erfüllt ist.
	
	Example:
	`if FileExist("xy.tpc5") = True then`
	`  BLOCK`
	`endif`

try-catch	Wird verwendet, um kritische Fehler abzufangen. Der Code-Teil in **block** wird ausgeführt und überwacht. Sollte dabei ein **Fehler auftreten**, welcher die Formel mit einer

Fehlermeldung stoppen würde, so wird der Code in **errorBlock** ausgeführt. Hier kann je nach dem eine detaillierte Meldung zum Fehler abgelegt werden oder auch ein Zähler mit der Anzahl der weiteren Versuche dekrementiert werden.

> **Achtung: Der im Formel Editor erstellte Code sollte generell auch ohne try-catch Statements lauffähig sein!**

Sauberer Code sollte auf Grenzfälle und Ausnahmen geprüft und diese falls möglich abgefangen werden. Grenzen von Arrays, Listen und Zähler sollten auch ohne diese try-catch Funktionen ohne Fehler lauffähig sein. Es gibt Ausnahmen auf welche man als Programmierer keinen Einfluss hat. Zum Beispiel das lesen von Dateien von einem Netzlaufwerk kann viele äussere Einflüsse haben: Netzwerkunterbruch, Verzögerungen im Netzwerk, Datei ist von einer anderen Applikation blockiert, etc. In solchen Fällen kann das try-catch Statement eine wertvolle Hilfe bieten.

Example:

```
; 1. Useful but more sophisticated example
; copy the latest BDF file from a TraNET FE device
; you may have to connect first with the file Explorer to its
IP address

noOfRetries=5
t_retry[s]=0.1
fNameDest = "myFile#.bdf"

for nbOfCopdyTry=1 to noOfRetries
  fileCopySuccessful=True
  try
    ;Path to the DAQ data directory
    tmp=GetInfo("url",0)
    tmp = SplitString(tmp, ":")
    urlDAQ = tmp(0)
    pathDAQ = PathCombine("\\", urlDAQ, "Measurement Data")
    fNameSource=GetNewestFile(pathDAQ,"*.bdf")
    fnameDestNew = FileNameInc(fNameDest)
    CopyFile(fNameSource, fnameDestNew,false)

  catch errorDict
    ; an error has occured
    fileCopySuccessful=False
    msg = errorDict("Message")
    srcPath = errorDict("SourcePath")
    dateTime = errorDict("DateTime")
    lineNr = errorDict ("LineNr")
    errorPos = errorDict ("Pos")
    errorLen = errorDict("Length")
    cancelled = errorDict("Cancelled")
    dateTimeString=StringFormat(dateTime,"")
    WriteLine("myLog.txt", Tabulator, msg, srcPath, dateTi-
meString, lineNr, errorPos, errorLen, cancelled, nbOfCopdy-
Try)
  endtry

  if fileCopySuccessful=True then
    exitloop
  endif
    delay(t_retry)
```


	`next`
	
	`; 2.) Easy to understand, but not so good example`
	`; create an array with 10 entries`
	`myArr = Array(0 to 9) as double`
	
	`try`
	`  val1 = myArr(4) ; no Error`
	`  val2 = myArr(12) ; will raise an Error`
	`catch errorDict`
	`  ; definde a value for val2`
	`  val2 = NotDefined`
	`  msg = errorDict("Message")`
	`  srcPath = errorDict("SourcePath")`
	`  dateTime = errorDict("DateTime")`
	`  lineNr = errorDict ("LineNr")`
	`  errorPos = errorDict ("Pos")`
	`  errorLen = errorDict("Length")`
	`  cancelled = errorDict("Cancelled")`
	`  dateTimeString=StringFormat(dateTime,"")`
	`  ; write a line into a Logfile`
	`  WriteLine("myLog.txt", Tabulator, msg, srcPath, dateTi-`
	`meString, lineNr, errorPos, errorLen, cancelled)`
	`endtry`

UnescapeString(text)	Die Funktion **UnescapeString** dient der einfacheren Formatierung von Texten. Der Parameter **text** ist ein String. Sonderzeichen wie Tabulatoren oder Zeilenumbrüche lassen sich so einfach in den Text einfügen. Häufig verwendete Escape Charakter sind Zeilenumbruch \r, \n oder Tabulator \t
	
	Example:
	`; \r\n creates a new line`
	`str =  UnescapeString("1st Row with text\r\n2nd Row with more`
	`text")`
	`Pause(str)`
	`; Same example without using function "UnexcapeString"`
	`str = "1st Row with text" + "\r"C + "\n"C + "2nd Row with more`
	`text"`
	`Pause(str)`
	
	`; Write a line to text file with tabulator separated`
	`str = UnescapeString("1st item\t2nd item\t3rd item")`
	`WriteLine("myText.txt", "", str)`
	`; This can also be written like this`
	`WriteLine("myText.txt", Tabulator, "1st item", "2nd item", "3rd`
	`item")`

Using(filepath)	Das Schlüsselwort **using** wird dazu verwendet, eine Funktionsdatei (*.fnc) filename einzubinden (Siehe Beispiel).
	
	Example:
	`; Create a file "MathOperators.fnc" with the following content:`
	
	`Function Addition(value0, value1)`
	`  Addition = value0 + value1`
	`EndFunction`
	
	`; Include the above file "MathOperators.fnc" into a new formula`
	`with the following statement:`
	`using "MathOperators.fnc"`
	
	`; The function Addition from "MathOperators.fnc" will be in-`
	`cluded because`
	`; of the usage of using.`

	`result = Addition(2, 3)`

VariableToString(a)	Liest den Namen einer Variablen.
	
	Der Parameter **a** ist eine im Formel Editor **definierte Variable**. Rückgabewert **value** ist vom Typ String zeigt den **Namen dieser Variable**.
	
	Siehe auch die Funktionen **GetVariableValue** und **SetVariable-Value**.
	
	Example:
	`myVariable = 20`
	`name = VariableToString(myVariable)`
	`Pause(name) ;myVariable`

While	Erstellt eine Schlaufe welche **block** so lange ausführt wie die Bedingung **i=b** richtig ist. Die Bedingung kann die folgenden Operatoren verwenden: "=", ">", "<", ">=", "<=", "<>". Mehrere Bedingungen können mit den Wörtern **and**, **or** und **not** zusammengehängt werden.
	
	Der Operator **<>** entspricht dem Ausdruck **not equal** (!=)
	
	Example:
	`i = 100`
	
	`do while i > 15`
	`  i = i - 1`
	`loop`

WriteForNext(key, value)	**WriteForNext** dient zusammen mit **ReadOfPrevious** als eine Art globaler Speicher. Die Variabeln sind für die nächste Formelberechnung erhältlich. Der Parameter **value** wird unter dem Namen **key** gespeichert.
	
	Für mehr Informationen und Beispiel siehe die Funktion **ReadOfPrevious**.
	
	Example:
	`; Averaging`
	`; get last trace`
	`mytrace = ReadOfPrevious("mytrace")`
	
	`; measure and smooth new trace`
	`tr = c0A1;`
	`tr = Smooth(tr, 100)`
	
	`; first run?`
	`if mytrace = NotDefined then`
	`  ; yes, first run`
	`  mytrace = tr`
	`else`
	`  ; no, not the first run`
	`  mytrace = (mytrace + tr) /2`
	`endif`
	
	`; store value for next calculation`
	`WriteForNext("mytrace",mytrace)`

34.15 Report Generator

XlCenter	Schlüsselwörter für den Parameter **alignment** der Funktionen **XlSetHeader** und **XlSetFooter**. **Example:** `XlSetHeader("report.xlsx", XlCenter, "Report")`

XlChangeColumn(cell, number)	Addiert die angegebene Zahl **number** zu der definierten Zelle **cell**. Dabei wird die Kolonne von **cell** beinflusst. **cell** muss ein String (z.B. "B5").
	
	Example:
	`cell = "B2"`
	`newCell = XlChangeColumn(cell, 2) ; "D2"`

XlChangeRow(cell, number)	Addiert die angegebene Zahl **number** zu der definierten Zelle **cell**. Dabei wird die Zeile (row) von **cell** beinflusst. **cell** muss ein String (z.B. "B5").
	
	Example:
	`cell = "B2"`
	`newCell = XlChangeRow(cell, 2) ; "B4"`

XlCreateEmptyFile()	Erstellt eine **leere** Excel Datei und gibt einen zufälligen Namen mit der Endung ".xlsx" als Rückgabewert zurück. Dieser Name muss weiterverwendet werden um auf der Datei arbeiten zu können. Per default ist dieser Datei der Pfad "data" des aktuellen Experiments zugewiesen. Diese Funktion empfiehlt sich, wenn errechnete Werte schnell in eine EXCEL-Datei gespeichert werden sollen. Sie beruht nicht auf einer zuvor erstellen Vorlage, kann aber im Nachhinein mit EXCEL weiterverarbeitet werden. Wenn diese Funktion aufgerufen wird, kann **XlOpenFile** entfallen.
	
	Example:
	`fnamework = XlCreateEmptyFile()  ; creates a random pattern file name`
	`XlSetCellValue(fnamework, "A10", 7.123) ; write some data`
	
	`fnnameFinal = "MyReport_#.xlsx" ; define a file name, # will be replace by an increasing number`
	`fnnameFinal = FileNameInc(fnnameFinal) ; increase counter and replace #`
	`XlSave(fnamework, fnnameFinal)  ; save the created ExcelSheet into the final File.`

XlGetCellReferences(filepath, searchKey [, all])	Die Funktion sucht in der ganzen EXCEL-Mappe **filepath** nach dem Schlüssel **searchKey** und gibt die gefundene Zellen-Bezeichnung zurück (z.B. "B5"). Diese Parameter müssen vom Typ String sein. Schlüsselwörter müssen natürlich vorher in der EXCEL-Mappe (Vorlage) eingetragen worden sein. Falls **keyword** nicht gefunden wird, bringt die Funktion "**n. def.**" zurück. Es empfiehlt sich Schlüsselwörter zu wählen, die sonst kaum vorkommen (z.B. "%V1"). Zwischen Klein- und Grossbuchstaben wird kein Unterschied gemacht. Falls der optionale Parameter **all** = **True** angegeben wird, wird ein Array bzw. eine Liste

	zurückgegeben. Darin sind alle Zellen-Bezeichnungen eingetragen, bei welchen das Schlüsselwort **searchKey** gefunden wurde.
	
	Example:
	`fn$ = ".\Excel_Templates\Template_1.xlsx"`
	`fnam$ = XlOpenFile(fn$)`
	`cellRef = XlGetCellReferences(fnam$, "%x")`

XlGetCellValue(filepath, cell)	Gibt den Wert in der Zelle **cell**. zurück. In der Regel ist **cell** eine einzelne Zelle ("B3"). Als **cell** wird ein String erwartet. Je nach dem was in der Zelle eingetragen ist, wird ein String oder eine Zahl zurückgegeben. Es kann aber auch ein Zellenbereich definiert werden (z.B. "A1:A4" oder "A1:D1"). Der Zellenbereich darf jeweils nur in derselben Kolonne oder Zeile definiert sein (nicht z.B. "A1:C3"). Auch muss vorher ein entsprechendes Array oder eine Liste deklariert worden sein. Es wird dann das Array oder die Liste abgefüllt mit den String- oder Zahlen-Werten der Zellen. Arrays müssen als String-Array oder Double-Array deklariert werden, je nachdem ob Strings oder Zahlen in den Zellen erwartet werden. Wenn eine Zelle leer ist, wird ein Leerstring bzw. "n. def." zurückgegeben. Wenn eine Liste deklariert wurde, enthalten danach deren Elemente Strings oder Zahlen.
	
	Example:
	`fn$ = ".\Excel_Templates\Template_1.xlsx"`
	`fnam$ = XlOpenFile(fn$)`
	`v1 = XlGetCellValue(fnam$, "A1") ; Einzelwert`
	`valArr$ = Array(0 to 3) as String`
	`valArr$ = XlGetCellValue(fnam$, "A2:A5")`
	`valArrD = Array(0 to 3) as Double`
	`valArrD = XlGetCellValue(fnam$, "B2:E2")`
	`valArrL = List()`
	`valArrL = XlGetCellValue(fnam$, "A2:E2")`

XlInitializeCOM()	Bei **XlInitializeCOM** muss eine gültige Version EXCEL ab 2007 installiert sein. Es wird eine Report-Mappe geöffnet die auch direkt während oder nach der Messung mit EXCEL bearbeitet werden kann. Bei **XlInitializeXML** wird keine Installation von EXCEL benötigt. Der erstellte Report wird durch eine XML-Datei (*.xlsx-Datei) gebildet. Diese kann danach weiterverwendet und mit EXCEL bearbeitet werden. Wenn keine der obigen Funktionen (XlInitializeCOM/XlInitializeXML) aufgerufen wird, verwendet der Formel Editor standardmässig XML Excel.
XlInitializeXML()	Bei **XlInitializeCOM** muss eine gültige Version EXCEL ab 2007 installiert sein. Es wird eine Report-Mappe geöffnet die auch direkt während oder nach der Messung mit EXCEL bearbeitet werden kann. Bei **XlInitializeXML** wird keine Installation von EXCEL benötigt. Der erstellte Report wird durch eine XML-Datei (*.xlsx-Datei) gebildet. Diese kann danach weiterverwendet und mit EXCEL bearbeitet werden.

	Wenn keine der obigen Funktionen (XlInitializeCOM/XlInitializeXML) aufgerufen wird, verwendet der Formel Editor standardmässig XML Excel.

XlInsertImage(filepath, name, cell [, width [, height]])	Fügt den Bildschirminhalt im Fenster **name** an der angegebenen Position **cell** in der EXCEL-Mappe **filepath** ein. Dies kann der **name** einer **einzelnen Kurvenanzeige**, einer **Skalar-Tabelle**, einer **ganzen Seite** oder einer **Bild-Datei** (inkl. Pfadangaben) sein. **cell** vom Typ String ist in der Regel eine einzelne Zelle (z.B. "B2"). Sie bestimmt die linke obere Ecke des einzufügenden Bildes. Für **cell** kann aber auch ein Bereich angegeben werden (z.B. "A4:F30"). Der Zellen-Bereich bestimmt dabei den Ort und die Grösse des eingefügten Bildes.
	Optional kann auch die Breite **width** und Höhe **height** des eingefügten Bildes definiert werden. Wird nur die Breite **width** angegeben, so wird die Höhe proportional dazu bestimmt (gleiche Seitenverhältnisse wie beim Original-Bild). **width** und **height** sind in Anzahl Pixels anzugeben und vom Typ Integer. Wenn ein Zellen-Bereich (nicht nur eine Einzel-Zelle) vorgegeben ist, wird **width** und **height**ignoriert.
	Die Einstellungen unter "Extras / Einstellungen / Benutzeroberfläche / Snapshot" Weisser Hintergrund" haben einen Einfluss auf die eingefügte Grafik.
	
	Example:
	`fn$ = ".\Excel_Templates\Template.xlsx"`
	`fnam$ = XlOpenFile(fn$)`
	`XlInsertImage(fnam$, "Waveform 1", "A4:E20")`
	`XlInsertImage(fnam$, "Skalar_A 1", "A21", 100, 300)`

XlLeft	Schlüsselwörter für den Parameter **alignment** der Funktionen **XlSetHeader** und **XlSetFooter** **Example:** `XlSetHeader("report.xlsx", XlLeft, "Report")`

XlOpenFile(filepath)	Öffnet eine bestehende Excel-Datei mit dem Namen **filename** (normalerweise eine Vorlage). Als Rückgabewert erhält man wieder den Dateinamen ergänzt mit dem absoluten Datei-Pfad. Dieser muss in nachfolgenden Excel-Funktionen verwendet werden.
	
	Hinweis: Wird Excel mit **XlInitializeXML** initialisiert, können nur ***.xlsx** Dateien verwendet werden. Bei **XlInitializeCOM** können auch ältere EXCEL-Dateiformate zur Anwendung kommen.
	
	Example:
	`XlInitializeXML()`
	`fn$ = ".\Excel_Templates\Template_1.xlsx"`
	`filename$ = XlOpenFile(fn$)`
	`XlSetCellValue(filename$, "A10", 7)`

XlRight	Schlüsselwörter für den Parameter **alignment** der Funktionen **XlSetHeader** und **XlSetFooter** **Example:**

	`XlSetHeader("report.xlsx", XlRight, "Report")`

XlSave(filepath [, newFilepath])	Speichert die offene EXCEL Datei **filepath** inkl. allen Änderungen.
	Wenn die Vorlage nicht überschrieben werden soll, muss für den zweiten Parameter **newFilepath** ein anderer Name (inkl. Pfad) gewählt werden.
	Solange danach der Befehl **CloseFile** nicht ausgeführt wird, bleibt die Excel-Mappe geöffnet.
	
	Example:
	`XlInitializeXML ( )`
	`fn$ = ".\Excel_Templates\Template_1.xlsx"`
	`fnam$ = XlOpenFile(fn$)`
	`fnRep$ = ".\Excel_Reports\Report_#.xlsx"`
	`; Block with value insertions in EXECL Sheet.`
	`XlSave (fn$, fnRep$) ; Saves the EXCEL file (incl. all insertions) in the Report File.`

XlSelectSheet(filepath, sheetname)	Wählt in der EXCEL-Mappe **filepath** das Tabellen-Blatt mit dem Name **sheetname**. Wenn dieser Befehl fehlt, wird standardmässig das erste Blatt gewählt.
	
	Example:
	`fn$ = ".\Excel_Templates\Template_1.xlsx"`
	`filename$ = XlOpenFile(fn$)`
	`XlSelectSheet(filename$, "Tabelle2")`

XlSetCellValue(filepath, cellOrKey, value)	Setzt in der Excel-Mappe **filepath** den Wert **value** in die Zelle **cellOrKey**. Anstelle einer Zelle (z.B. "B2") kann auch ein Schlüssel **cellOrKey** vorgegeben werden (dadurch kann **XlGetCellReferences** entfallen).
	Wenn **value** ein Array, Liste oder eine Kurve ist, werden die Werte standardmässig in die **Kolonne** der spezifizierten Zelle **cell** eingefügt. Möchte man die Werte in eine **Zeile** einfügen, muss die Zelle folgendermassen definiert werden:**"A7:row".**
	Kurven (wie auch Arrays oder Listen) dürfen nicht zu lang sein für die Übertragung in Excel. Die maximale Länge (Anzahl Samplings, bzw. Elemente) beträgt: Übertragung in eine **Kolonne**: 1'048'577 - Linie des ersten Wertes (z.B. <=1'048'571, wenn **cell** = "x6 "). Übertragung in eine **Zeile**: 16'385 - Kolonne des ersten Wertes(z.B. <=116'382, wenn **cell** = "Cx:row "). Zu lange Kurven (grosse Blocklängen) können vorher mit den Funktionen **Slice**, **Resampling** oder **Skip** verkürzt werden.
	
	Example:
	`filename = "excel\Template.xlsx"`
	`filename = XlOpenFile(filename)`
	`colArr = Array(1,2,3,4)`
	`XlSetCellValue(filename, "A1", colArr)`
	`XlSetCellValue(filename, "A1:row", colArr)`

XlSetFooter(filepath, alignmentOrKey, value)	Setzt in der **Fusszeile** der Excel-Mappe **filepath** den Wert **value** ein. Der Parameter **alignmentOrKey** bestimmt die Position an welcher **value** eingefügt werden soll. Dieser kann ein Schlüsselwort **XlLeft, XlCenter, XlRight** (für Links, Zentriert, Rechts) oder ein Suchwort sein. Ein Suchwort muss vorher in der Kopf- bzw. Fuss-Zeile der Excel-Vorlage eingefügt worden sein.

	Example:
	`Fn$ = "excel\Template.xlsx"`
	`Fn$ = XlOpenFile(Fn$)`
	`Skey$="%Hd-1" ; Should be set in Template`
	`XlSetHeader(Fn$, XlLeft, "Report")`
	`XlSetHeader(Fn$, Skey$, "Unit Test 1")`
	`XlSetFooter(Fn$, XlCenter, "Result Table")`

XlSetHeader(filepath, alignmentOrKey, value)	Setzt in der **Kopfzeile** der Excel-Mappe **filepath** den Wert **value** ein. Der Parameter **alignmentOrKey** bestimmt die Position an welcher **value** eingefügt werden soll. Dieser kann ein Schlüsselwort **XlLeft, XlCenter, XlRight** (für Links, Zentriert, Rechts) oder ein Suchwort sein. Ein Suchwort muss vorher in der Kopf- bzw. Fuss-Zeile der Excel-Vorlage eingefügt worden sein.
	
	Example:
	`Fn$ = "excel\Template.xlsx"`
	`Fn$ = XlOpenFile(Fn$)`
	`Skey$="%Hd-1" ; Should be set in Template`
	`XlSetHeader(Fn$, XlLeft, "Report")`
	`XlSetHeader(Fn$, Skey$, "Unit Test 1")`
	`XlSetFooter(Fn$, XlCenter, "Result Table")`

34.16 Signal Analysis

<table>
<tr><td>a(index*)</td><td>Gibt den Wert an Position index aus a zurück. Mehrere Indizes werden in mehrdimensionalen Arrays verwendet.

Der Parameter index ist eine Zeit (Sekunden) wenn a eine Kurve ist.

Example:
<pre>tr1[V] = Sinus(10, 50, 0, 1E3, 1E3) ; 10V amplitude (20Vpp), 50Hz
t1 = 0.022
val1[V] = tr1(t1) ; val1=5.8779

SetCrs("Waveform 1","A",t1); set cursor A to position of t1</pre></td></tr>
<tr><td>Aic(a [, start, end])</td><td>Wird für Acoustic Emission (AE) -Anwendungen verwendet. Algorithmus zur automatischen Erkennung der ersten Bewegung der AE basierend auf dem AIC-Picker.

Gemäss Wikipedia EN: The Akaike information criterion (AIC) is an estimator of the relative quality of statistical models for a given set of data a. Given a collection of models for the data, AIC estimates the quality of each model, relative to each of the other models. Thus, AIC provides a means for model selection. AIC is founded on information theory: it offers an estimate of the relative information lost when a given model is used to represent the process that generated the data. (In doing so, it deals with the trade-off between the goodness of fit of the model and the simplicity of the model.)

Die Parameter start und end können verwendet werden um die Berechnung auf eine spezifische Länge einzuschränken.

Example:
<pre>; AIC Example
tr = GenerateSignal()
SamplingRate = 1/TSample(tr)
triggerTime = TTrigger(tr) - TBegin(tr)

; generate an AIC trace
; the AIC array has to be converted back to a trace
trAic = ConvToTrace(Aic(ConvToArray(tr)), SamplingRate)
; set trigger time according source trace
trAic = SetTrigger(trAic, triggerTime)

; search minimum of AIC Trace and set cursor to its position
tmin = TOfMin(trAic)
SetCrs("Waveform 2","A",tmin)

; create a test signal
Function GenerateSignal()
    tr1 = Noise(2,1E6,1E3)
    tr2 = tr1 *2
    trNew = Merge(tr1, tr2)

    t1 = TBegin(trNew)
    t2 = TEnd(trNew)
    ; place trigger to the last quarter of the signal
    trNew = SetTrigger(trNew,(t2-t1)*0.75)</pre></td></tr>
</table>

	`GenerateSignal = trNew` `EndFunction`

Area(a [, start, end])	Berechnet die Fläche von **a** relativ zum Nullpunkt. Werte grösser als null werden positiv, kleiner als null negativ gewertet. Die optionalen Parameter **start** und **end**, vergleichbar mit der Funktion **Slice** von Kurven, definiert die Start- und Endzeit der Berechnung.
	
	Siehe auch die Funktion **Int** für die Rückgabe von Kurven.
	
	Example:
	`; create a signal v[m/s] with 10[m/s^2] acceleration`
	`v[m/s] = Ramp(10,1E3,10E3)`
	`a[m/s^2] = diff(v)`
	
	`s_trace = Int(v)  ; creates a trace`
	`s_num  = Area(v)  ; numeric result`

DutyCycle(a [, start, end [, level, hysteresis]])	Berechnet das Verhältnis von positiver Pulsbreite zur Periodendauer von **a**. Der Rückgabewert ist eine Zahl. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll. Weitere optionale Parameter sind **level** und **hysteresis**. Sind diese nicht definiert, wird **level** auf den Mittelwert des Spitzenwerts gelegt, **hysteresis** beträgt 5% des Spitzenwerts.
	
	Example:
	`tr = sigRectangle()  ; create a rectangle signal`
	
	`dutC  = DutyCycle(tr)  ; dutC = 25.275`
	`pwNeg = PulseWidthNeg(tr) ; pwNeg = 136us`
	`pwPos = PulseWidthPos(tr) ; pwPos = 46us`
	
	`EndFormula`
	
	`Function sigRectangle()`
	`  ; Create a rectangle signal with a duty cycle of 25%`
	`  periodCount = 10 ; 3 periods`
	`  fs = 1e6 ; 1 MHz sampling rate`
	`  freqRect[Hz] = 5.5e3 ; 5.5 kHz signal frequency`
	`  dt = 1/freqRect`
	`  peak[V] = 3.3 ; 3.3V peak value`
	`  dutyCyclePer[%] = 25 ; 25% duty cycle`
	`  listPair = List()`
	`  listPair() = 0`
	`  listPair() = peak`
	`  listPair() = dt * dutyCyclePer / 100`
	`  listPair() = peak`
	`  listPair() = listPair(2) + 1/fs`
	`  listPair() = 0`
	`  listPair() = listPair(4) + dt * (100 - dutyCyclePer) / 100 - 1/fs`
	`  listPair() = 0`
	`  sigRectangle = CreateSignal(periodCount, fs, listPair)`
	`EndFunction`

Energy(a [, start, end])	Berechnet das Integral der quadrierten Punkte von **a**. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll. **Example:**

	`trace = c0A1` `E = Energy(trace)`

FindEvent(a, type, start, end, level, hysteresis)	Die Funktion kann dazu verwendet werden um Flanken in **a** zu detektieren. Der Parameter **type** ist eine Zahl (0 … 2) und definiert auf welchen Flankentyp gesucht werden soll:
	
	• Positiv = 0
	• Negativ = 1
	• Beide = 2
	
	Alle anderen Werte führen zu einem Fehler in der Formel.
	
	Die Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll. Parameter **level** definiert den gesuchten Wert und **hysteresis** kann zur besseren Detektion bei Rauschen verwendet werden. Generell sollte Hysteresis nicht auf null gesetzt werden.
	
	FindEvent ist eine Erweiterung der Funktion **TCross**.
	
	Example:
	`trace = Sinus(10, 50E-3, 0, 1E3, 30E3)`
	
	`t1 = TBegin(trace)`
	`t2 = tEnd(Trace)`
	
	`; Slopes: 0=positive, 1=negative, 2=both`
	`type = 0`
	`tEvent1 = FindEvent(trace, type, t1, t2, 2, 0.1) ; tEvent1 = 0.64094`
	`SetCrs("Waveform 1","A", tEvent1)`
	
	`; Slopes: 0=positive, 1=negative, 2=both`
	`type = 1`
	`tEvent2 = FindEvent(trace, type, t1, t2, 2, 0.1) ; tEvent2 = 9.3591`
	`SetCrs("Waveform 1","B", tEvent2)`

Freq(a [, level, hysteresis])	Der Parameter **a** muss eine periodische Kurve sein. Die Funktion **Freq** berechnet aus den Nulldurchgängen der Kurve die **mittlere Frequenz**. Das Resultat **variable** ist eine Zahl in Hz. Mit den optionalen Parametern **level** und **hysteresis** kann die Nulllinie und die Hysterese angepasst werden.
	
	Example:
	`; create a signal`
	`tr = Sinus(10,50,0,1E3,1E3)`
	`tr = tr + Noise(2,tr)`
	
	`; calculate frequency`
	`f = Freq(tr) ; f = 50Hz`

Frequency(a [, start, end [, level, hysteresis]])	Berechnet die Frequenz von **a**. Der Rückgabewert ist eine Zahl. Um sinnvolle Werte zu erhalten sollte **a** periodisch sein. Der Parameter **start** und **end** sind Zeitmarken. Zwischen diesen wird die Frequenz evaluiert. Die Parameter **level** und **hysteresis** sind optional. Sind diese nicht definiert, wird **level** auf den

Mittelwert des Spitzenwerts gelegt, **hysteresis** beträgt 5% des Spitzenwerts.

> Diese Funktion ist ähnlich wie **Freq**. Die Funktion **Freq** ist aus Rückwärtskompatibilität noch erhältlich.

Example:

```
; create a signal
tr = Sinus(10,50,0,1E3,1E3)
tr = tr + Noise(2,tr)

; define start and end
t1 = GetCrs("Waveform 1","A")
t2 = GetCrs("Waveform 1","B")

; calculate frequency
f = Frequency(tr,t1, t2) ; f = approx. 50Hz
```

GetBlockIndexes(a, axisMode, start, end)

Die Funktion **GetBlockIndexes** gibt alle erhältlichen Block Indizes zwischen **start** und **end** zurück.
Rückgabewert **variable** ist ein Array mit dem Start- und Endindex.
Der Parameter **axisMode** definiert, wie die Werte von **start** und **end** zu interpretieren sind. Mögliche Werte sind:

- 0 = Relative Time since Start
- 1 = Absolute Time
- 2 = Relative Time (Zero at Trigger)
- 3 = Samples

Die beiden Parameter **start** und **end** sind entweder die **Zeitmarken in Sekunden**, eine Absolute Zeit vom Typ **DateTime**, oder die Position in **Samples**.

> Diese Funktion macht am meisten Sinn, wenn mehrere Blöcke in **a** vorliegen.

Example:

```
; multiblock recording, approx. 20 blocks for this example
; place cursors somewhere within the first and the last block

UseMemory(True)
tr = c0A1

; read position of cursors
t1 = GetCrs("Waveform 1","A")
t2 = GetCrs("Waveform 1","B")

; get first and last selected block
crsBlocks = GetBlockIndexes(tr,0,t1,t2)

; make sure position 0 is the smaller block number
if crsBlocks(0) > crsBlocks(1) then
  tmp = crsBlocks(0)
  crsBlocks(0) = crsBlocks(1)
  crsBlocks(1) = tmp
endif


averagedFFTCuve = NotDefined
```

```
SetReference("ref1",true,tr) ; workaround to read one block
for blkIndex = crsBlocks(0) to crsBlocks(1)
  blk = ref1.blkIndex  ;read one block

  ; calculate averaged FFT of selected blocks
  traceFFT = FFT(blk)
  if averagedFFTCuve = NotDefined then
    averagedFFTCuve = ConvToSpectrumTrace(traceFFT)
  else
    averagedFFTCuve = (ConvToSpectrumTrace(traceFFT) + avera-
gedFFTCuve * blkIndex) / (blkIndex + 1)
  endif

next
```

GetDate([a])

Die Funktion **GetDate** liefert Datum und Zeit vom Computer zurück. Falls der optionale Parameter **a** definiert wurde, wird Datum und Uhrzeit von **a** zurückgegeben. Der Rückgabewert **variable** ist eine Zahl im Format double, welche Datum- und Zeit-Angaben enthält.

Das Formelbeispiel zeigt wie aus der zurückgegebenen Zahl ein Datum und Uhrzeit extrahiert wird.

Example:

```
date = GetDate(c0A1)

date = date / 10E3
year = Integer(date)
date = date - year
date = date * 100
month = Integer(date)
date = date - month
date = date * 100
day = Integer(date)
date = date - day
date = date * 100
hour = Integer(date)
date = date - hour
date = date * 100
minute = Integer(date)
date = date - minute
date = date * 100
sec = Integer(date)
```

GetDateTime([a])
GetDateTime(text, format)

Gibt das aktuelle Datum und Systemzeit zurück. Mit dem optionalen Parameter **a** wird Datum und Zeit des Triggerzeitpunkt der Kurve **a** zurückgegeben.

Konvertiert einen Datumsstring **text** mit Datum- und Zeitinformationen in eine DateTime Variabel. Der Parameter **format** definiert wie der Datumsstring zu interpretieren ist.

Example:

```
computerDateTime = GetDateTime()
traceDateTime = GetDateTime(c0A1)
dateTime3 = GetDateTime("12.01.2017 13:15:00", "MM.dd.yyyy
HH:mm:ss")
```


GetNrOfSamples(a, start, end)	Die Funktion gibt die Anzahl Samples von **a** zwischen den Zeitmarken **start** und **end** zurück. **start** und **end** sind in Sekunden definiert.
	
	Wenn **end < start** ist, wird ein **negativer** Wert zurückgegeben.
	
	Example:
	`; Read the number of samples from a trace`
	`tr = c0A1`
	`t0 = TBegin(tr)`
	`t1 = TEnd(tr)`
	`n1 = GetNrOfSamples(tr, t0, t1)`
	
	`; use the position of Cursors`
	`t0 = GetCrs("Waveform 1","A")`
	`t1 = GetCrs("Waveform 1","B")`
	`n2 = Abs(GetNrOfSamples(tr, t0, t1))`

GetTimeSpan(seconds [, minutes [, hours [, days]]])	Konvertiert die Parameter **seconds, minutes, hours days** in einen TimeSpan. Eine TimeSpan Variabel ist lediglich eine Zeitdifferenz. Der Parameter **seconds** ist zwingend, die restlichen Parameter sind optional.
	
	Example:
	`tsp1 = GetTimeSpan(100)  ; 00:01:40  -> 1min, 40sec`
	`tsp2 = GetTimeSpan(100, 4) ; 00:05:40  -> 5min, 40sec`
	`tsp3 = GetTimeSpan(86400)  ; 1.00:00:00 -> 1day`

GetTimeValues(a)	Konvertiert den Parameter **a** in einen Dictionary, welcher einfachen Zugriff auf Datums- und Zeitinformationen gibt.
	
	String **Schlüssel** von Dictionary für **DateTime**:
	"day", "month", "year", "second", "minute", "hour"
	
	String **Schlüssel** von Dictionary für **TimeSpan**:
	"seconds", "minutes", "hours", "days", "totalMilliseconds", "totalSeconds", "totalMinutes", "totalHours", "totalDays"
	
	Example:
	`; DateTime varibale`
	`t1 = GetDateTime()`
	`dictDateTime = GetTimeValues(t1)`
	`year = dictDateTime("year")`
	
	`Delay(1)`
	
	`; TimeSpan variable`
	`tDelta = GetDateTime()-t1`
	`dictTimeSpan = GetTimeValues(tDelta)`
	`milis = dictTimeSpan("totalMilliseconds")`

GetUnit(a)	Liefert den Namen der physikalischen Einheit von **a**. Rückgabewert **variable** ist vom Typ String. Siehe auch die Funktion **SetUnit**

	Example:
	`; create a signal`
	`tr = Sinus(10,50,0,10E3,1E3)`
	`tr[Bar] = tr + Noise(2,tr)`
	
	`; filtering keeps the unit`
	`tr2 = LowPass(tr,Bessel,4,100)`
	
	`; calculation removes the unit`
	`tr3 = tr2 *1`
	
	`; restore origin unit`
	`Unit = GetUnit(tr)`
	`SetUnit(tr3, Unit)`

Max(a [, start, end])	Die Funktion sucht den **grössten Wert** von **a**.
	Die optionalen Parameter **start** und **end grenzen den Suchbereich** ein.
	
	Das Resultat **variable** ist eine Zahl.
	
	Mit der verwendung von **start** und **end** können Daten reduziert und die Berechnung somit auch **beschleunigt werden**.
	
	Example:
	`; example with trace`
	`tr = Sinus(10,50,0,1E3,1E3)`
	`tr = tr + Noise(2,tr)`
	`trMax = Max(tr)`
	`trMin = Min(tr)`
	
	`; example with list`
	`myList = List()`
	`myList() = 3`
	`myList() = -2`
	`myList() = 8`
	`myList() = -10`
	
	`listMax = Max(myList)`
	`listMin = Min(myList)`

Mean(a)	Die Funktion bildet den **arithmetischen Mittelwert** von **a**. Das Resultat **variable** ist eine Zahl.
	
	Example:
	`; create a sinewave signal`
	`tr = Sinus(10,50,0,1E3,1E3)`
	`tr += 5`
	
	`trmean = Mean(tr) ; trmean = 5`

MeanP(a)	Die Funktion bildet den **arithmetischen Mittelwert** von **a**. Die Kurve **a** sollte periodisch sein. Das Resultat **variable** ist eine Zahl.
	
	Example:
	`fs = 1e6`
	`offset = 1.55`
	`noiseTrace = Noise(0.01, fs, 1e4)`
	`sineTrace = Sinus(1, 1e3, 0, fs, 1e4) + noiseTrace + offset`
	
	`; Evaluate the given offset`
	`evaluatedOffset = MeanP(sineTrace)`

Min(a [, start, end])	Die Funktion sucht den **kleinsten Wert** von **a**.
	Die optionalen Parameter **start** und **end grenzen den Suchbereich** ein.
	
	Das Resultat **variable** ist eine Zahl.
	
	Mit der verwendung von **start** und **end** können Daten reduziert und die **Berechnung somit auch beschleunig** werden.
	
	Example:
	`; example with trace`
	`tr = Sinus(10,50,0,1E3,1E3)`
	`tr = tr + Noise(2,tr)`
	`trMax = Max(tr)`
	`trMin = Min(tr)`
	
	`; example with list`
	`myList = List()`
	`myList() = 3`
	`myList() = -2`
	`myList() = 8`
	`myList() = -10`
	
	`listMax = Max(myList)`
	`listMin = Min(myList)`

Overflow(a [, start, end])	Signalisiert einen Überlauf des ADCs beim Signal **a**.
	
	Die optionalen Parameter **start** und **end** sind Zeitmarken in Sekunden, zwischen denen die Berechnung ausgeführt werden soll.
	
	Der Rückgabe wert ist nicht definiert (NaN), wenn kein Überlauf gefunden wurde. Ansonsten ist dieser eine Zeitmarke vom Typ Double.
	
	Example:
	`tr = c0A1`
	`of = OverFlow(tr)`
	`uf = UnderFlow(tr)`
	
	`if (of = NotDefined)and(uf = NotDefined) then`
	`  SetMessage("Signals within Range", true)`
	`else`
	`  Pause("Check Range settings!")`
	`endif`

OverShotNeg(a [, start, end])	Gibt die Stärke einer Unterschwingung (negative Überschwingung) in % von **a** zurück. Die optionalen Parameter **start** und **end** sind Zeitmarken in Sekunden, zwischen denen die Berechnung ausgeführt werden soll. Rückgabewert ist die Unterschwingung in Prozent. Bei Analyse von periodischen Signalen kann es dazu führen, dass keine brauchbare Auswertung vorgenommen werden kann und der Rückgabewert nicht definiert ist (NaN).

	Example:
	`trace = c0A1`
	`t1 = GetCrs("Waveform 1","A")`
	`t2 = GetCrs("Waveform 1","B")`
	
	`oShot = OvershotNeg(trace,t2, t1)`
	
	`if oShot <> NotDefined then`
	`  ; your code here`
	`endif`

OvershotPos(a [, start, end])	Gibt die Stärke einer Überschwingung (positive Überschwingung) in % von **a** zurück.
	
	Die optionalen Parameter **start** und **end** sind Zeitmarken in Sekunden, zwischen denen die Berechnung ausgeführt werden soll.
	
	Rückgabewert ist die Überschwingung in Prozent. Bei Analyse von periodischen Signalen kann es dazu führen, dass keine brauchbare Auswertung vorgenommen werden kann und der Rückgabewert nicht definiert ist (NaN).
	
	Example:
	`trace = c0A1`
	`t1 = GetCrs("Waveform 1","A")`
	`t2 = GetCrs("Waveform 1","B")`
	
	`oShot = OvershotPos(trace,t2, t1)`
	
	`if oShot <> NotDefined then`
	`  ; your code here`
	`endif`

PeakPeak(a [, start, end])	Berechnet die Differenz zwischen Maximum und Minimum von **a**.
	
	Die optionalen Parameter **start** und **end** sind Zeitmarken in Sekunden, zwischen denen die Berechnung ausgeführt werden soll.
	
	Example:
	`tr = Sinus(10,50,0,10E3,1E3)`
	`pp1 = PeakPeak(tr) ; pp1 = 20`
	
	`; PeakPeak() = Max() - Min()`
	`pp2 = Max(tr)-Min(tr) ; pp2 = 20`

Period(a [, start, end [, level, hysteresis]])	Sucht auf der Basislinie des Signals **a** die Durchgänge und berechnet daraus die mittlere Periodendauer.
	
	Die optionalen Parameter **start** und **end** sind Zeitmarken in Sekunden, zwischen denen die Berechnung ausgeführt werden soll. Weitere optionale Parameter sind **level** und **hysteresis**. Sind diese nicht definiert, wird **level** auf den Mittelwert des Spitzenwerts gelegt, **hysteresis** beträgt 5% des Spitzenwerts.
	
	Example:
	`tr = Sinus(10,50,0,10E3,1E3)`
	`tper = Period(tr) ; tper = 20ms`

PulseWidthNeg(a [, start, end])

Berechnet die negative Pulsbreite beim Pegel der Basisline (Mittel von Maximum - Minimum) von **a**. Sind mehrere Perioden im Signal, wird die mittlere Pulsbreite berechnet. Rückgabewert ist eine Zahl. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll.

Example:

```
tr = sigRectangle()  ; create a rectangle signal

dutC  = DutyCycle(tr)     ; dutC = 25.275
pwNeg = PulseWidthNeg(tr) ; pwNeg = 136us
pwPos = PulseWidthPos(tr) ; pwPos = 46us


EndFormula

Function sigRectangle()
  ; Create a rectangle signal with a duty cycle of 25%
  periodCount = 10 ; 3 periods
  fs = 1e6 ; 1 MHz sampling rate
  freqRect[Hz] = 5.5e3 ; 5.5 kHz signal frequency
  dt = 1/freqRect
  peak[V] = 3.3 ; 3.3V peak value
  dutyCyclePer[%] = 25 ; 25% duty cycle
  listPair = List()
  listPair() = 0
  listPair() = peak
  listPair() = dt * dutyCyclePer / 100
  listPair() = peak
  listPair() = listPair(2) + 1/fs
  listPair() = 0
  listPair() = listPair(4) + dt * (100 - dutyCyclePer) / 100 -
1/fs
  listPair() = 0
  sigRectangle = CreateSignal(periodCount, fs, listPair)
EndFunction
```

PulseWidthPos(a [, start, end])

Berechnet die positive Pulsbreite beim Pegel der Basislinie (Mittel von Maximum - Minimum) der Kurve **trace**. Sind mehrere Perioden im Signal, wird die mittlere Pulsbreite berechnet. Rückgabewert ist ein Skalar vom Typ Double. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll.

Example:

```
tr = sigRectangle()  ; create a rectangle signal

dutC  = DutyCycle(tr)     ; dutC = 25.275
pwNeg = PulseWidthNeg(tr) ; pwNeg = 136us
pwPos = PulseWidthPos(tr) ; pwPos = 46us


EndFormula

Function sigRectangle()
  ; Create a rectangle signal with a duty cycle of 25%
  periodCount = 10 ; 3 periods
  fs = 1e6 ; 1 MHz sampling rate
  freqRect[Hz] = 5.5e3 ; 5.5 kHz signal frequency
  dt = 1/freqRect
  peak[V] = 3.3 ; 3.3V peak value
  dutyCyclePer[%] = 25 ; 25% duty cycle
  listPair = List()
  listPair() = 0
```

	`listPair() = peak`
	`listPair() = dt * dutyCyclePer / 100`
	`listPair() = peak`
	`listPair() = listPair(2) + 1/fs`
	`listPair() = 0`
	`listPair() = listPair(4) + dt * (100 - dutyCyclePer) / 100 - 1/fs`
	`listPair() = 0`
	`sigRectangle = CreateSignal(periodCount, fs, listPair)`
	`EndFunction`

RectifiedMean(a [, start, end])	Berechnet den gleichgerichteten Mittelwert des Signals **a**. Rückgabewert ist eine Zahl. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll.
	
	Example:
	`tr1 = Sinus(100, 50,  0, 1E3, 1E3) ; 10V amplitude -> 20Vpp`
	
	`vrms = RMS(tr1)  ; 70.711V`
	`vrec = RectifiedMean(tr1) ; 63.138V`
	
	`; Ration between RMS and Rectified Mean is defined as approx. 1.1`
	`ratio = vrms/vrec ;  1.1199`

RectifiedMeanPeriodic(a [, start, end])	Berechnet den gleichgerichteten Mittelwert des Signals **a**. Diese Funktion sucht die Nulldurchgänge der Basislinie und berechnet den gleichgerichteten Mittelwert ganzer Perioden. Rückgabewert ist eine Zahl. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll.
	
	Example:
	`tr1 = Sinus(100, 50,  0, 1E3, 1E3) ; 10V amplitude -> 20Vpp`
	
	`vrms = RMSP(tr1)  ; 70.675`
	`vrec = RectifiedMeanPeriodic(tr1) ; 63.073`
	
	`; Ration between RMS and Rectified Mean is defined as approx. 1.1`
	`ratio = vrms/vrec ;  1.1205`

RiseFallTime(a, startLevel, errorLevel [, start, end])	Ermittelt die Anstiegs- oder Abfall-Zeit von **a** bei dem Startpegel **startLevel** und Fehlerpegel **errorLevel**. Die Parameter **startLevel** und **errorLevel** sind beide in Prozent. Der Rückgabewert ist eine Zahl welche die Zeit in Sekunden darstellt. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll.
	
	Example:
	`tr = Ramp(0,10,1E3,1E3)`
	
	`t1 = RiseFallTime(tr,10,90) ; t1 = 0.8s`
	`s1 = Slope(tr) ; s1 = 10.01`

RMS(a [, start, end])	Berechnet den **Effektivwert** von **a**. Es wird dabei nicht auf ganze Perioden geachtet. Das Resultat **variable** ist eine Zahl. Die beiden optionalen Parametern **start** und **end** sind Zeitwerte, innerhalb dieser wird **RMS** berechnet. **Example:** `tr = Sinus(10,50,0,1E3,1E3)`

	`t1 = GetCrs("Waveform 1","A")`
	`t2 = GetCrs("Waveform 1","B")`
	
	`valRMS  = RMS( tr, t1, t2) ; exact between t1 and t2`
	`valRMSP = RMSP(tr, t1, t2) ; check for whole periods between t1 and t2`

RMSP(a [, start, end])	Berechnet den **Effektivwert** über eine oder mehrere **ganze Perioden** von **a**. Das Resultat **variable** ist eine Zahl. Die beiden optionalen Parametern **start** und **end** sind Zeitwerte, innerhalb dieser wird **RMSP** berechnet. Basisline wir automatisch ermittelt.
	
	Example:
	`tr = Sinus(10,50,0,1E3,1E3)`
	`t1 = GetCrs("Waveform 1","A")`
	`t2 = GetCrs("Waveform 1","B")`
	
	`valRMS  = RMS( tr, t1, t2) ; exact between t1 and t2`
	`valRMSP = RMSP(tr, t1, t2) ; check for whole periods between t1 and t2`

SetDate(a, b)	Die Funktion setzt das aktuelle Datum und Uhrzeit (Computer) von **a**. Der optionale Parameter **b** kann verwendet werden um das Datum und Uhrzeit einer **Kurve** oder **Zahl** zu verwenden. Das Resultat **variable** ist wieder eine Kurve.
	
	GetDate zeigt wie eine Zahl das Datum und Uhrzeit repräsentieren kann.
	
	Example:
	`; crate a trace, wait some seconds and create a second trace`
	`tr1 = Sinus(10,50,0,1E3,1E3)`
	`Delay(2)`
	`tr2 = Sinus(12,50,0,1E3,1E3)`
	
	`; calculate time difference`
	`deltatime1 = GetDate(tr2) - GetDate(tr1)`
	
	`; adjust date of tr2 to the one of tr1`
	`tr2 = SetDate(tr2, tr1)`
	`; difference will be zero now`
	`deltatime2 = GetDate(tr2) - GetDate(tr1)`

SetDateTime(a, date)	Setzt ein neues Datum **date** der der Kurve **trace**.
	
	Example:
	`dateTimeNow = GetDateTime()`
	`signal = c0A1`
	`signal = SetDateTime(signal, dateTimeNow)`

SettlingTime(a, startLevel, endLevel [, start, end])	Gibt die Zeit zurück bis das Signals **trace** innerhalb eines Fehlerbandes bleibt. Startwert (0%) ist der Beginn der Kurve, der Endwert (100%) das Ende der Kurve. Das Fehlerband bezieht sich auf den Endwert. **startlevel** und **endlevel** sind Werte in Prozent. Das Fehlerband definiert sich als ±(Endwert-**endlevel**). Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll.
	
	Example:
	`trace = c0A1`
	`tSettling = SettlingTime(trace, 10, 90)`

SetUnit(a, unit)	Setzt die physikalische Einheit von **a** auf den Namen des Parameters **unit**.
	
	Siehe auch die Funktion **GetUnit**.
	
	Example:
	`; create a signal`
	`tr = Sinus(10,50,0,10E3,1E3)`
	`tr[Bar] = tr + Noise(2,tr)`
	
	`; filtering keeps the unit`
	`tr2 = LowPass(tr,Bessel,4,100)`
	
	`; calculation removes the unit`
	`tr3 = tr2 *1`
	
	`; restore origin unit`
	`Unit = GetUnit(tr)`
	`SetUnit(tr3, Unit)`

Slope(a [, start, end])	Berechnet die Steigung von **a**. Slope ist definiert als dy/dx. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll. Rückgabewert ist eine Zahl mit der Einheit [Messeinheit pro Sekunde].
	
	Example:
	`tr = Ramp(0,10,1E3,1E3)`
	
	`t1 = RiseFallTime(tr,10,90) ; t1 = 0.8s`
	`s1 = Slope(tr) ; s1 = 10.01`

SlopeLinearRegression(a [, start, end])	Berechnet die Steigung von **a**, führt zusätzlich zur Funktion Slope noch eine Lineare Regression durch. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll. Rückgabewert ist eine Zahl mit der Einheit [Messeinheit pro Sekunde].
	
	Example:
	`trace = c0A1`
	`sl = SlopeLinearRegression(trace)`

StdDevPeriodic(a [, start, end])	Sucht die Durchgänge der Basisline von **a** und berechnet die Standardabweichung ganzer Perioden. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll. Rückgabewert **variable** ist eine Zahl.
	
	Example:
	`trace = c0A1`
	`dev = StdDevPeriodic(trace)`

TBegin(a)	Die Funktion liefert den Zeitpunkt (X-Achse) des ersten Samples von **a** in Sekunden zurück. Der Rückgabewert **variable** ist eine Zahl. Im Scope-Mode ist die Zeit relativ zum Triggerzeitpunkt, bei Aufnahmen im Multiblock- Continuous-

oder ECR-Mode ist die Zeit Relativ zum Zeitpunkt des Startkommandos der Aufnahme.

Example:

```
; generate a signal
tr = Noise(10,1E6,1E6)

; get TBegin and TEnd
t1 = TBegin(tr)
t2 = TEnd(tr)

; set cursors to its position
SetCrs("Waveform 1","A", t1)
SetCrs("Waveform 1","B", t2)
```

TCross(a, start, end, level)

Die Funktion sucht beginnend von **start** in Richtung **end** (geht vorwärts und rückwärts) die nächste Stelle, bei der **a** den Wert **level** über- oder unterschreitet. Das Resultat **variable** ist eine Zahl in Sekunden. Die Funktion gibt **NotDefined** zurück falls keine Über- oder Unterschreitung gefunden wurde.

Siehe auch die Funktion **EnEvents**.

Example:

```
; generate a signal
tr = Sinus(10,50,0,1E3,1E3)
tr = tr + Noise(2,tr)

; crossing level
level = 0.5

; create a list with all crossings
levelList = List()

t1 = TBegin(tr)
t2 = TEnd(tr)
ts = TSample(tr)
tpos = t1

; start from beginning to the end of the trace
do while tpos < t2
    tpos = TCross(tr, tpos, t2, level)
    levelList() = tpos  ; add to list
    tpos = tpos+ts        ; move one sample forward
loop
```

TEnd(a)

Die Funktion liefert den Zeitpunkt (X-Achse) des letzten Samples von **a** in Sekunden zurück. Der Rückgabewert **variable** ist eine Zahl.

Im Scope-Mode ist die Zeit relativ zum Triggerzeitpunkt, bei Aufnahmen im Multiblock- Continuous- oder ECR-Mode ist die Zeit Relativ zum Zeitpunkt des Startkommandos der Aufnahme.

Example:

```
; generate a signal
tr = Noise(10,1E6,1E6)

; get TBegin and TEnd
t1 = TBegin(tr)
t2 = TEnd(tr)
```

	`; set cursors to its position`
	`SetCrs("Waveform 1","A", t1)`
	`SetCrs("Waveform 1","B", t2)`

TOfMax(a [, start, end])	Die Funktion gibt die Zeit (Sekunden) vom Maximum zurück welches in **a** gefunden wurde. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll.
	
	Bei Aufnahmen im Multiblock- Continuous- oder ECR-Mode ist die Zeit Relativ zum Zeitpunkt des Startkommandos der Aufnahme.
	
	Example:
	`; generate a signal`
	`tr = Noise(10,1E6,1E6)`
	
	`; read TOfMax and TOfMin`
	`tmax = TOfMax(tr)`
	`tmin = TOfMin(tr)`
	`t1`
	
	`; set cursors to its position`
	`SetCrs("Waveform 1","A",tmax)`
	`SetCrs("Waveform 1","B",tmin)`

TOfMin(a [, start, end])	Die Funktion gibt die Zeit (Sekunden) vom Minimum zurück welches in **a** gefunden wurde. Die optionalen Parameter **start** und **end** sind Zeitmarken, zwischen denen die Berechnung ausgeführt werden soll.
	
	Bei Aufnahmen im Multiblock- Continuous- oder ECR-Mode ist die Zeit Relativ zum Zeitpunkt des Startkommandos der Aufnahme.
	
	Example:
	`; generate a signal`
	`tr = Noise(10,1E6,1E6)`
	
	`; read TOfMax and TOfMin`
	`tmax = TOfMax(tr)`
	`tmin = TOfMin(tr)`
	
	`; set cursors to its position`
	`SetCrs("Waveform 1","A",tmax)`
	`SetCrs("Waveform 1","B",tmin)`

TSample(a)	Die Funktion liefert die **Abtastrate** (Zeit in Sekunden) zurück.
	
	Die Abtastrate ist der Kehrwert der Abtastfrequenz (T = 1/f oder auch f=1/T).
	
	Example:
	`tr = Sinus(10,50,0,1E3,1E3)`
	
	`t1 = TSample(tr) ; t = 0.001s  = 1ms,  interval between each sample`
	`f1 = 1/t1  ; f = 1'000Hz = 1kHz, sample rate frequency`

TTrigger(a)	Die Funktion liefert den Trigger-Zeitpunkt (X-Achse) der Kurve **trace** in Sekunden zurück.

	Bei Aufnahmen im Scope- oder Continuous-Mode ist das Resultat gleich 0. Bei Aufnahmen im Multiblock- oder ECR-Mode ist die Zeit Relativ zum Zeitpunkt des Startkommandos der Aufnahme.
	Example:
	`; Multi Block recording`
	`tr = c0A1.1  ; second block`
	`t1 = TTrigger(tr)`
	
	`tr = c0A1.4  ; fifth block`
	`t2 = TTrigger(tr)`
	
	`; create a list with all trigger time of all blocks`
	`mylist = List()`
	`for each item in EnBlocks(c0A1, false)`
	`  mylist() = TTrigger(item)`
	`next`

UnderFlow(a [, start, end])	Signalisiert einen Unterlauf des ADCs beim Signal **a**.
	
	Die optionalen Parameter **start** und **end** sind Zeitmarken in Sekunden, zwischen denen die Berechnung ausgeführt werden soll.
	
	Der Rückgabe wert ist nicht definiert (NaN), wenn kein Unterlauf gefunden wurde. Ansonsten ist dieser eine Zeitmarke vom Typ Double.
	
	Example:
	`tr = c0A1`
	`of = OverFlow(tr)`
	`uf = UnderFlow(tr)`
	
	`if (of = NotDefined)and(uf = NotDefined) then`
	`  SetMessage("Signals within Range", true)`
	`else`
	`  Pause("Check Range settings!")`
	`endif`

34.17 Signal Generations

CreateSignal(periodCount, fs, (t1, y1)+) **CreateSignal(periodCount, fs, a)** **CreateSignal(periodCount, trace, (t1, y1)+)** **CreateSignal(periodCount, trace, a)**	Mit **CreateSignal** können beliebige Signal-Kurven erzeugt werden. Der Parameter **periodCount** gibt die Anzahl der Perioden (Wiederholungen) an. Dieser Wert wird jeweils auf eine ganze Zahl abgerundet und sollte somit >=1 sein. **fs** definiert die Abtastrate (in Hz) der zu erzeugenden Kurve. **(t1, y1)+** besteht immer aus zwei Zahlenwerten: Der erste für den Zeitpunkt auf der X-Achse, der zweite für den Amplitudenwert auf der Y-Achse. Die Anzahl der x-y-Wertepaare sind nicht limitiert. Mit **CreateSignal** können beliebige Signal-Kurven erzeugt werden. Der Parameter **periodCount** gibt die Anzahl der Perioden (Wiederholungen) an. Dieser Wert wird jeweils auf eine ganze Zahl abgerundet und sollte somit >=1 sein. **fs** definiert die Abtastrate (in Hz) der zu erzeugenden Kurve.

	Der Parameter **a** ist eine Aufzählung (Array, Liste) welche immer zuerst mit einem x-Wert (Zeit) beginnt, gefolgt von einem y-Wert.
	
	Mit **CreateSignal** können beliebige Signal-Kurven erzeugt werden. Der Parameter **periodCount** gibt die Anzahl der Perioden (Wiederholungen) an. Dieser Wert wird jeweils auf eine ganze Zahl abgerundet und sollte somit >=1 sein. Der Parameter **trace** setzt alle Einstellungen für die X-Achse (Abtastrate, etc.).
	
	(t1, y1)+ besteht immer aus zwei Zahlenwerten: Der erste für den Zeitpunkt auf der X-Achse, der zweite für den Amplitudenwert auf der Y-AChse. Die Anzahl der x-y-Wertepaare sind nicht limitiert.
	
	Mit **CreateSignal** können beliebige Signal-Kurven erzeugt werden. Der Parameter **periodCount** gibt die Anzahl der Perioden (Wiederholungen) an. Dieser Wert wird jeweils auf eine ganze Zahl abgerundet und sollte somit >=1 sein. Der Parameter **trace** setzt alle Einstellungen für die X-Achse (Abtastrate, etc.).
	
	Der Parameter **a** ist eine Aufzählung (Array, Liste) welche immer zuerst mit einem x-Wert (Zeit) beginnt, gefolgt von einem y-Wert.
	
	Example:
	`; Create a triangle signal`
	`periodCount = 5 ; 5 periods`
	`fs = 1e6 ; 1 MHz sampling rate`
	`dt = 1/fs`
	`freqTriangle[Hz] = 1e3 ; 1 kHz signal frequency`
	`peak[V] = 1 ; 1V peak value`
	`sigTriangle = CreateSignal(periodCount, fs, 0, -peak, 1/fre-`
	`qTriangle, peak)`
	
	`; Create a rectangle signal with a duty cycle of 25%`
	`periodCount = 3 ; 3 periods`
	`fs = 1e6 ; 1 MHz sampling rate`
	`freqRect[Hz] = 5.5e3 ; 5.5 kHz signal frequency`
	`dt = 1/freqRect`
	`peak[V] = 3.3 ; 3.3V peak value`
	`dutyCyclePer[%] = 25 ; 25% duty cylce`
	`listPair = List()`
	`listPair() = 0`
	`listPair() = peak`
	`listPair() = dt * dutyCyclePer / 100`
	`listPair() = peak`
	`listPair() = listPair(2) + 1/fs`
	`listPair() = 0`
	`listPair() = listPair(4) + dt * (100 - dutyCyclePer) / 100 -`
	`1/fs`
	`listPair() = 0`
	`sigRectangle = CreateSignal(periodCount, fs, listPair)`

Noise(ampl, fs, length) **Noise(ampl, trace)**	Die Funktion erzeugt ein **Rausch-Signal**. **ampl**, **fs** (in Hz), und **length** (in Samples) müssen Zahlen sein (Gauss-verteiltes Zufalls-Signal mit Standardabweichung **ampl**).

	Die Funktion erzeugt ein **Rausch-Signal**. Der Parameter **ampl** setzt die Amplitude des Rauschens. **trace** definiert die **Abtastrate** und **Länge** der Rauschkurve. **Example:** `tr = Noise(5, 1E3, 100E3)`

Ramp(slope, fs, length)
Ramp(slope, trace)
Ramp(y0, y1, trace)
Ramp(y0, y1, fs, length)
	Die Funktion erzeugt eine **Rampe** mit der Steilheit **slope** (Einheit pro Sekunde). **slope**, **samplerate** (in Hz) und **length** (in Samples) müssen Zahlen sein.
	
	Erstellt eine **Rampe** mit der Steilheit **slope** (Einheit pro Sekunde). Der Parameter **trace** definiert die **Abstastrate** und **Länge** von der Rampe.
	
	Erstellt eine **Rampe** mit dem Startpunkt **y0** und Endpunkt **y1**. Die Zeit zwischen **y0** und **y1** ist definiert bei der **Abtastrate** (fs) und **Länge** von **trace** (Zeit=Länge/fs).
	
	Erstellt eine **Rampe** mit dem Startpunkt **y0** und Endpunkt **y1**. Die Zeit zwischen **y0** und **y1** ist definiert durch den Parameter **fs** und **length** (Zeit=length/fs).
	
	Example:
	`; creat a sawtooth signal`
	`tr = Ramp(0,1,1E3,1E3)`
	`trst = Merge(tr,tr,tr,tr)`
	
	`; create a signal v[m/s] with 10[m/s^2] acceleration`
	`v[m/s] = Ramp(10,1E3,10E3)`
	`a[m/s^2] = diff(v)`

Sinus(ampl, freq, phase, fs, length)
Sinus(ampl, freq, phase, trace)
	Die Funktion gibt eine **sinusförmige Kurve** zurück. Die Parameter **ampl** (in V oder Unit), **freq** (in Hz), **phase** (in °), **fs** (in Hz), und **length** (Anzahl Smaples) müssen Zahlen sein. Peak-Peak Werte der Amplitude ist das zweifache des Parameters **ampl**.
	
	Die Funktion gibt eine **sinusförmige Kurve** zurück. Die Parameter **ampl** (in V oder Unit), **freq** (in Hz), **phase** (in °), **samplerate** (in Hz), und **length** (Anzahl Smaples) von **trace**. Peak-Peak Werte der Amplitude ist das zweifache des Parameters **ampl**.
	
	Example:
	`tr1 = Sinus(10, 50,  0, 1E3, 1E3) ; 10V amplitude -> 20Vpp`
	`tr2 = Sinus( 8, 50, 180, tr1)  ; 180° phase shift to tr1`

34.18 Signal Processing

Averaging	Schlüsselwort für **Type** bei der Funktion **DataReduction**.

Convolution(a, b)	Erstellt eine Faltung der beiden Arrays **a** and **b**. Der Rückgabewert ist ein Array mit dem gewichteten Durchschnitt von Array **a**. Die Grösse des neuen Arrays ist die von **a + b - 1**.
	
	Example:
	`a = Array(1:3) ; a= 1, 2, 3`
	`b = Array(2:4) ; b= 2, 3, 4`
	`c = Convolution(a,b) ; c= 2, 7, 16, 17, 12`

	`len = Length(c); len= 5`

Correlation(a, b)	Berechnet die Kreuzkorrelation zwischen **a** und **b**. Falls **a** und **b** gleich sind entspricht dies einer Autokorrelations-Funktion.
	
	Example:
	`; Find a spike pattern in a sinus signal with the help of cross correlation.`
	`; The spike pattern can be of any complexity. For this example, it is only`
	`; a simple spike.`
	`; ----------`
	`fs = 1e6`
	`dt = 1/fs`
	`traceLength = 1e4`
	`peak = 2.5`
	`peakLength = dt * 5`
	`tSpikeStart = Random(0, dt * traceLength - peakLength) ; Randomly generate a time where the spike starts`
	`sineTrace = Sinus(1, 1e3, 0, fs, traceLength) ; Generate a sinus signal`
	`spikeTrace = CreateSignal(1, fs, 0, 0, tSpikeStart, 0, tSpikeStart + peakLength/2, peak, tSpikeStart + peakLength, 0, dt * traceLength, 0) ; Create a spike signal`
	`sineWithSpike = sineTrace + spikeTrace + Noise(0.1, fs, traceLength) ; Add the spike and noise to the sinus signal`
	`spikeOnly = Slice(sineWithSpike, tSpikeStart - peakLength, tSpikeStart + 2*peakLength) ; Get only the spike with the help of the slice function`
	`xyCorr = Correlation(sineWithSpike, spikeOnly) ; Find the displacement of the spike within the sinus signal`
	`tSpike = TOfMax(xyCorr) - traceLength * dt ; Calculate the position of the spike begin`
	
	`; Find the phase shift of two signals with cross correlation`
	`; ----------`
	`fs = 1e6`
	`freqSin = 1e3`
	`dt = 1/fs`
	`traceLength = 1e4`
	`sineTrace1 = Sinus(1, freqSin, 0, fs, traceLength) ; Generate sine trace 1`
	`sineTrace2 = Sinus(1.45, freqSin, 30, fs, traceLength) ; Generate sine trace 1 with 30° phase shift`
	`xyCorrPhase = Correlation(sineTrace1, sineTrace2) ; Correlate both signals to find the time of maximum match`
	`tPhaseShift[s] = TOfMax(xyCorrPhase) - traceLength * dt ; Calculate the position of the spike begin`
	`tPhaseInDegree[°] = 360 / (1/freqSin) * (tPhaseShift) ; Calculate the phase shift in degrees`

DataReduction(a, type, factor)	Die Datenmenge der Kurve **a** wird verringert um den Parameter **factor**. **factor** ist eine Zahl welche >= 1 sein muss. Die Abtastrate wir automatisch angepasst. Für den Parameter **type** sind folgende Werte gültig: • **Skipping**; Jedes n-te Sample wird verwendet für die Resultatskurve. • **Averaging**: Der gleitende Mittelwert wird für n-Samples berechnet und als einen Wert in die Resultatskurve abgespeichert. • **MinMaxEnvelope** erzeugt eine Zick-Zack-Kurve. D.h. es werden Wertepaare erzeugt, zuerst der Minimal- dann

	der Maximal-Wert innerhalb des Reduktions-Bereichs. Die Sampling-Zeit wird entsprechend angepasst.
	
	Example:
	`tr1 = c0A1 ; get a trace`
	`TrRed=DataReduction(tr1, Averaging,10)`

ExponentialFit(a [, returnCoeffs [, coeff_a, coeff_b]])	Die Funktion **ExponentialFit** erzeugt eine Exponential-Kurve welche optimal zur Kurve **a** passt. Bei einer sinkenden Kurve im positiven Bereich oder steigender Kurve im negativen Bereich geht die Funktion davon aus, dass **a-->0 bei x-->+**. Wenn die Verhältnisse umgekehrt sind wird davon ausgegangen, dass **a-->0 bei x--> -**.
	
	Der Parameter **returnCoeffs** ist optional. Wird dieser auf **True** gesetzt, erhält man anstelle einer Kurve ein Array mit den Koeffizienten y0 und x0, gem. f(x) = y0 * e x0*x CoefArr(0)=y0 (Amplituden-Wert bei x=0) CoefArr(1)=x0
	
	Aus x0 kann die Zeitkonstante ermittelt werden (T = 1/x0).
	
	Die beiden optionalen Parameter **coeff_af** und **coeff_b** können als Startparameter zur Optimierung des Fittings verwendet werden.
	
	Example:
	`; create a test signal`
	`tr = Ramp(0,10,1E3,1E3)`
	`tr = tr + noise(5,tr)`
	
	`; trace`
	`trExp = ExponentialFit(tr, false)`
	
	`; array`
	`trArr = ExponentialFit(tr, true)`
	`T = 1/trArr(1)`

FrqDemod(a [, level [, hysteresis]])	Die Funktion sucht Nulldurchgänge von **a** und berechnet daraus an den einzelnen Stellen die Frequenz. Mit den **optionalen** Parametern **level** und **hysteresis** kann die Basislinie und die Hysterese angepasst werden. Diese Werte bestimmen dann den Suchalgorithmus für die Nulldurchgänge. Als Rückgabewert erhält man wieder eine Kurve.
	
	Example:
	`; create a signal`
	`tr1 = Sinus(10,50,0,1E3,1E3)`
	`tr2 = Sinus(10,100,0,1E3,1E3)`
	`tr = MergeTraces(tr1,tr2,tr1)`
	
	`trFreq= FrqDemod(tr)`

GetXAxisMode(name)	Gibt die aktuelle Einstellung der X-Achse des Wavefrom **name** zurück. Rückgabewerte für **variable** sind: 0 = Relative time since Start 1= Absolute Time 2= Relative Time (Zero at Trigger) 3= Samples **Example:**

	`; 0 = Relative time since Start`
	`; 1 = Absolute Time`
	`; 2 = Relative Time (Zero at Trigger)`
	`; 3 = Samples`
	`mode = GetXAxisMode("Waveform 1")`

MinMaxEnvelope	Schlüsselwort für **Type** bei der Funktion **DataReduction**.

Phase(a, b [, level_a, hysteresis_a, level_b, hysteresis_b])	Die Funktion sucht Nulldurchgänge von **a** und **b**, berechnet dann aus beiden die **Phasenverschiebung** und gibt diese als Kurve in Grad zurück.
	
	Es wird der Versatz von **a** in Bezug auf **b** als Referenz verglichen.
	
	Optional können noch Level (**level_a**, **level_b**) und Hysterese (**hysteresis_a**, **hysteresis_b**) der beiden kurven definiert werden um den Algorithmus für die Nulldurchgänge zu beeinflussen.
	
	Example:
	`tr1 = Sinus(10,50, 0,1E3,1E3)`
	`tr2 = Sinus(10,50,20,1E3,1E3) ; tr2 starts with 20° offset`
	
	`; compare tr2 to tr1`
	`ph = Phase(tr2, tr1) ; curve rises up to 20°`

RegressionPoly(a, order [, returnCoeffs [, tExtrapolAhead [, tExtrapolBehind]]]) **RegressionPoly(a, fs, order, returnCoeffs [, tExtrapolAhead [, tExtrapolBehind]])**	Die Funktion berechnet eine gefittete Kurve anhand von **a** welche der Polynom-Funktion n-ten Grades (**order**) entspricht (z.B. Polynome mit n=2 ==> (k1*a^2 + k2*a + k3). k1, k2, und k3 werden durch entsprechende Regression ermittelt). Die Resultatskurve weist keine grossen Sprünge und Unstetigkeiten mehr auf. Wenn der optionale dritte Parameter **returnCoeffs** auf **True** (=1) gesetzt wird, liefert die Funktion ein **Array** mit den Polynom-Koeffizienten k0, k1, k2, ... anstelle einer Kurve zurück. Die weiteren zwei Parameter **tExtrapolAhead** und **tExtrapolBehind** bestimmen, wieviele Sekunden vor und nach der berechneten Regression hinzugefügt werden sollen. Die Funktion berechnet eine gefittete Kurve anhand von **a** welche der Polynom-Funktion n-ten Grades (**order**) entspricht (z.B. Polynome mit n=2 ==> (k1*a^2 + k2*a + k3). k1, k2, und k3 werden durch entsprechende Regression ermittelt). **a** muss ein Array oder Liste sein. Der Parameter **fs** wird als Abtastrate für die Resultatskurve verwendet. Die Resultatskurve weist keine grossen Sprünge und Unstetigkeiten mehr auf. Wenn der optionale dritte Parameter **returnCoeffs** auf **True** (=1) gesetzt wird, liefert die Funktion ein **Array** mit den Polynom-Koeffizienten k0, k1, k2, ... anstelle einer Kurve zurück.

	Die weiteren zwei Parameter **tExtrapolAhead** und **tExtrapolBehind** bestimmen, wieviele Sekunden vor und nach der berechneten Regression hinzugefügt werden sollen.
	
	Example:
	`; create a test signal`
	`tr = Ramp(0,10,1E3,1E3)`
	`tr = tr + noise(5,tr)`
	
	`trReg = RegressionPoly(tr,3,false, 0.1, 0.05) ; add 100ms before und 50ms after the calculated trace`

Resampling(a, fs)	Ändert die Abtastrate der Kurve **a** auf die Frequenz **fs** (in Hz). Ist **fs** grösser als die ursprüngliche Abtastrate von **a**, werden die fehlenden Punkte linear interpoliert, iIst sie kleiner, werden Punkte entfernt.\
	
	Möglicherweise wird dadurch der Triggerzeitpunkt leicht verschoben, da dieser nach der Operation auf das nächst gelegene Sample gesetzt wird.
	
	Example:
	`tr_orig = c0A1`
	`OrigSR = 1/TSample(tr_orig)`
	`NewSR= 3.1234 * OrigSR`
	`tr_new = Resampling(tr_orig, NewSR)`

SetTrigger(a, t)	Die Funktion setzt den Trigger-Zeitpunkt **t** (Sekunden) der Kurve **a**. Das Verhalten ist vergleichbar mit einer manipulierten **Triggerverzögerung**. Die Werte für **TBegin** und **TEnd** bleiben auf den ursprünglichen Werten stehen.
	
	Der Rückgabewert **variable** ist wieder eine Kurve.
	
	Example:
	`tr = Sinus(10,50,0,1E3,1E3)`
	`trNew = SetTrigger(tr, 0.03 )`

SetTSample(a, t)	Die Funktion liefert die Kurve **trace** mit der Abtastperiode **t** (Sekunden) zurück. Die Zeitachse der Signalkurve wird verändert. Der Rückgabewert **variable** ist wieder eine Kurve. Diese Funktion ändert die Abtastrate. Die so modifizierte Kurve hat die gleiche Anzahl Samples, der Zeitbereich ist aber gestreckt oder gestaucht.
	
	Example:
	`trOrig = Sinus(10,50,0,1E3,1E3)`
	
	`tsNew = TSample(trOrig)/2`
	`trNew = SetTSample(trOrig, tsNew)`

SetXAxis(a, refTrace) **SetXAxis(a, t0 [, dt [, triggerSample [, unit]]])**	Erzeugt eine Kurve mit den Amplituden-Werten von **a** und den Zeitachsen-Werten der Kurve **refTrace** (Abtastrate, Trigger-Delay, Beginn, Ende usw.). Werte am Anfang und am Ende der Resultatkurve werden, je nach Überlappung abgeschnitten oder mit Null aufgefüllt.

	Falls die Abtastraten von **refTrace** unterschiedlich ist, erfolgt ein Resampling.
	
	Erzeugt eine Kurve mit den Amplituden-Werten von **a** und den folgenden Zeitachsenparametern:
	
	• **t0**: Begin der Resultatskurve
	• **dt**: Abtastrate der Resultatskurve
	• **triggerSample**: Effektives Trigger Sample der Resultatskurve
	• **unit**: Einheit der Resultatskurve
	
	Falls die Abtastraten **dt** sich von **a** unterscheidet, erfolgt ein Resampling.
	
	Example:
	`y = File("measure.tpc5", 0)`
	`x = c0A3.0`
	`z = SetXAxis(y, x)`

SetXAxisMode(name, axisMode)	Setzt den Axis Mode einer vorhandenen Kurvenanzeige **name**.
	
	Der zweite Parameter **axisMode** kann die folgenden Werte haben:
	
	• 0 = Relative Time since Start
	• 1 = Absolute Time
	• 2 = Relative Time (Zero at Trigger)
	• 3 = Samples
	
	Example:
	`; set axis mode to Relative Time since Start`
	`SetXAxisMode("Waveform 1", 0)`

Shift(a, b)	Schiebt eine Kurve **a** um den Wert **b** welcher entweder eine Kurve, eine Zeit (Sekunden) oder eine Kurvenanzeige sein kann. Der Startzeitpunkt (TBegin) wird neu gesetzt. Die Verschiebung wird nicht auf ganze Samplings gerundet. Von Beginn her betrachtet liegen die Stützwerte der neuen und alten Kurve also nicht mehr übereinander. Dies gilt es zu beachten, wenn bei weiteren Auswerte-Operationen der Beginn und nicht der Trigger-Zeitpunkt als Referenz genommen wird. Falls anstelle **b** eine zweite Kurve angegeben wird, so schiebt sich die Resultat-Kurve in den Zeitbereich von **b**. **Die Trigger der beiden Kurven stehen dann übereinander.
	
	Example:
	`tr = Sinus(10,50,0,1E3,1E3)`
	`trShift = Shift(tr, 0.5) ; 0.5 seconds to the right`

Skip(a, number)	Diese Funktion bewirkt ein **Resampling** von **a**, sodass die Resultatkurve entsprechend weniger oder mehr Abtastwerte (Downsampling oder Upsampling) aufweist. Falls **number** eine

	positive Zahl ist entsteht eine Resultatskurve mit **number** mal **weniger** Abtastwerten. Bei **number** als negative Zahl werden **Abs(number)** mal **mehr** Samplings erzeugt.
	
	Der Parameter **number** kann auch eine gebrochene Zahl sein. Zwischenwerte werden durch lineare Interpolation ermittelt.
	
	Diese Funktion, eventuell kombiniert mit Filtern, erlaubt eine Datenreduktion die dann sinnvoll ist, wenn Signale mit Überabtastung aufgezeichnet wurden und eine rechenintensive Nachauswertung (z.B. die Funktion **Correlation** oder **FFT**) durgeführt werden soll.
	
	Der Rückgabewert **variable** ist eine Kurve.
	
	Example:
	`tr = Sinus(10,50,0,1E3,1E3)`
	
	`trLess = Skip(tr,  2) ; half of the samples`
	`trMore = Skip(tr, -5) ; 5 time more samples`

Skipping	Schlüsselwort für **Type** bei der Funktion **DataReduction**.

Slice(a, start, end [,y0 [,y1]])	Die Funktion **schneidet den Bereich** zwischen **start** bis **end** von **a** aus. Die Parameter **start** und **end** werden in Sekunden angegeben wenn **a** eine Kurve ist. Ansonsten sind **start** und **end** wie ein Index zu behandeln.
	
	Ist **start < TBegin(a)** bzw. **end > TEnd(a)** wird die Kurve an der entsprechenden Stelle verlängert, falls mindestens **y0** angegeben wird. Falls nur **y0** angegeben ist, wird die Kurve **a** am Anfang und/oder Ende mit dem konstanten Amplitudenwert **y0** verlängert. Falls auch **y1** angegeben wird, verlängert sich die Kurve mit dem Amplitudenwert **y1** am Ende (falls **end > TEnd(a)**) und **y0** am Anfang (falls **start < TBegin(a)**). Das Resultat ist eine verkürzte oder verlängerte Kurve.
	
	Die **Slice** Funktion kann auch auf ein **Array** oder eine Liste **List** angewendet werden.
	
	Example:
	`variable = Slice(a, t0, t1)`
	
	`variable = Slice(a, t0, t1 [,y0 [,y1]])`
	`variable = Slice(Array or List, startIndex, endIndex)`

Spline(a, order, fs [, tExtrapolAhead[, tExtrapolBehind]]) **Spline(a, order, refTrace [, tExtrapolAhead[, tExtrapolBehind]])**	Berechnet einen Polynomzug aus **a**. Wenn **a** ein Array oder Liste ist, müssen diese Zeit-Wertepaare beinhalten (Zeit in Sekunden). Der Parameter **order** ist zurzeit fix auf 3 (cubic spline) gesetzt, andere Werte werden ignoriert. Der Parameter **fs** ist die Abtastrate (Frequenz) der Resultatkurve. **tExtrapolAhead** und **tExtrapolBehind** sind zwei optionale Zeit-

	Parameter (positive Werte, in Sekunden) welche die Kurve am Anfang oder Schluss verlängert.
	
	Berechnet einen Polynomzug aus **a**. Wenn **a** ein Array oder Liste ist, müssen diese Zeit-Wertepaare beinhalten (Zeit in Sekunden).
	Der Parameter **order** ist zurzeit fix auf 3 (cubic spline) gesetzt, andere Werte werden ignoriert.
	Der Parameter **refTrace** ist eine andere Kurve welche alle X-Achsen Informationen setzt (Abtastrate, Trigger Sample, TBegin, Absolutzeit, Einheit).
	tExtrapolAhead und **tExtrapolBehind** sind zwei optionale Zeit-Parameter (positive Werte, in Sekunden) welche die Kurve am Anfang oder Schluss verlängert.
	
	Example:
	`; create a triangle by x,y value pairs`
	`arr = Array(0,0, 0.05,0, 0.1,1, 0.2,0, 0.25,0)`
	`tr = CreateSignal(1, 1e3, arr)`
	`tr_sp = Spline(arr, 3, 1e3, 0.02, 0.015)`

StdDev(a, width)	Die Funktion berechnet die Standardabweichung aus **a**. Der Parameter **width** definiert wieviele Werte für die Berechnung berücksichtigt werden sollen.
	
	Das Resultat **variable** ist wieder eine Kurve, ein Array oder eine Liste.
	
	Example:
	`; create some random values`
	`; e.g. time duration for a light barrier in milliseconds`
	`items = 100`
	`myArr = Array(0 to items-1) as double`
	`for a = 0 to items-1 step 1.0`
	`  myArr(a) = random(58, 74)  ; random values form 58ms to 74ms`
	`next`
	
	`tmp = StdDev(myArr, items)  ; StdDev returns an array`
	`standardDeviation = tmp(items-1)  ; use last item`
	`meanValue = mean(myArr)  ; calculate mean`
	
	`; calculate the percentage of light barrier values below 68 ms`
	`pcVal = NormDist(68, meanValue, standardDeviation, true)`
	
	`; calculate the minimal duration to be on the upper 75%`
	`msVal = NormInv(0.25,meanValue,standardDeviation)`

Variance(a, width)	Die Funktion berechnet die Varianz aus **a**. Der Parameter **width** definiert wieviele Werte für die Berechnung berücksichtigt werden sollen.
	
	Das Resultat **variable** is wieder eine Kurve, ein Array oder eine Liste.
	
	Example:
	`tr = Sinus(10,50,0,1E3,1E3)`
	`arr = Array(0:99)`
	
	`varTr  = mean(Variance(tr,20)) ; a number instead of a trace`

	`varArr = (Variance(arr,10))(0) ; the first element of the array`

34.19 Spectrum (FFT)

AcousticAnalysis(a, doTerzAnalysis [, normedFrequencies])	Akustische Analyse von **a**. Der Rückgabewert ist ein Array vom Typ Double. Bei einer Oktaven Analyse sind 10, bei einer Terz (1/3 Oktave) Analyse 30 Werte im Rückgabe Array **variable**. Diese entsprechen der Darstellung der FFT-Anzeige.
	Die 31 Bänder (n+1) einer Terz-Analyse (1/3 Oktave) in Hz: 22.5, 28.2, 35.5, 45, 56, 71, 90, 112, 140, 180, 224, 280, 355, 450, 560, 710, 890, 1120, 1400, 1800, 2240, 2800, 3550, 4500, 5600, 7100, 9000, 11200, 14130, 17780, 22400
	Die 10 Bänder (n+1) einer Oktave-Analyse in Hz: 22.5, 45, 90, 180, 355, 710, 1400, 2800, 5600, 11200, 22400
	Die Spektrum Kurve **a** muss evtl. zuvor mit der Funktion **FFT** aus einer bestehenden Messung berechnet werden. Der Parameter **doTerzAnalysis** ist vom Typ **Boolean**. Ist dieser auf True wird eine Terz (1/3 Oktave), bei False eine Oktaven Analyse durchgeführt.
	Der Parameter **normedFrequencies** dient dazu eigene Frequenzbänder zur Analyse zu definieren.
	
	Example:
	`; create a signal`
	`tr1 = Sinus(10,1E3,0,1E6,1E3)`
	`tr2 = Noise(10,tr1)`
	`tr = MergeTraces(tr1, tr2, tr1)`
	
	`; create a FFT array`
	`arrFFT = fft(tr, HammingWin)`
	
	`; convert FFT array to spectrum trace`
	`trSpec = ConvToSpectrumTrace(arrFFT)`
	
	`; Terz Analysis, array with 30 items (1/3 octave)`
	`trAc = AcousticAnalysis(trSpec, true)`

Angle(a)	Berechnet den Winkelanteil von komplexen Zahlen. Das Resultat wird eine einzelne Zahl oder Double Array sein. Siehe auch **Abs, Conjugate, Real, Imag.**
	
	Example:
	`x = -3+5j ; complex number`
	
	`re = Real(x)  ; re = -3`
	`im = Imag(x)  ; im = 5`
	`ar = angle(x)  ; ar = 120.96°`
	`co = Conjugate(x) ; co = -3-5j`
	
	`; alternatively to  function angle()`
	`deg = atan(imag(x),real(x)) ; deg = 120.96°`

BlackmanHarris3Win	Schlüsselwort für den Parameter **window** der Funktion **FFT**, definiert das **Bewertungsfenster** beim Ausführen der Fast Fourier Transformation.
	
	Example:
	`sineTrace = Sinus(1, 1e3, 0, 1e6, 1e4)`
	`fftVal = FFT(sineTrace, BlackmanHarris3Win)`

BlackmanHarris4Win	Schlüsselwort für den Parameter **window** der Funktion **FFT**, definiert das **Bewertungsfenster** beim Ausführen der Fast Fourier Transformation.
	
	Example:
	`sineTrace = Sinus(1, 1e3, 0, 1e6, 1e4)`
	`fftVal = FFT(sineTrace, BlackmanHarris4Win)`

BlackmanWin	Schlüsselwort für den Parameter **window** der Funktion **FFT**, definiert das **Bewertungsfenster** beim Ausführen der Fast Fourier Transformation.
	
	Example:
	`sineTrace = Sinus(1, 1e3, 0, 1e6, 1e4)`
	`fftVal = FFT(sineTrace, BlackmanWin)`

Conjugate(a)	Mathematische Funktion für komplexe Zahlen. Invertiert den imaginären Anteil der einzelnen Elemente in **a**. Das Resultat wird wieder eine einzelne komplexe Zahl oder ein Array aus komplexen Zahlen sein. Siehe auch **Abs, Angle, Real, Imag**
	
	Example:
	`x = -3+5j ; complex number`
	
	`re = Real(x)  ; re = -3`
	`im = Imag(x)  ; im = 5`
	`ar = angle(x)  ; ar = 120.96°`
	`co = Conjugate(x) ; co = -3-5j`
	
	`; alternatively to  function angle()`
	`deg = atan(imag(x),real(x)) ; deg = 120.96°`

ConvToSpectrum-Trace(a)	Damit wird ein Array mit komplexen Werten **a** in eine Spektrum-Kurve gewandelt, Diese kann anschliessend in einem FFT-Kurvenanzeige dargestellt werden. Die resultierende Spektrum-Kurve ist als Spitzenwert (Peak) skaliert.
	
	Example:
	`tr = c0A1`
	`arrfft = FFT(tr, HannWin)`
	`trfft = ConvToSpectrumTrace(arrfft)`

FFT(a [, window])	Schnelle Fourier Transformation (Fast Fourier Transformation). Diese wird verwendet um **a** vom Zeitbereich in ein Array mit komplexen Werten im Frequenzbereich zu wandeln. Es werden alle Werte von Beginn bis am Ende von **a** gewandelt. Allenfalls kann die Kurve mit der Funktion **Slice** verkürzt werden.
	Der Parameter **window** ist die Fensterfunktion zur Gewichtung des Signals im Zeitbereich. Wenn dieser fehlt, wird automatisch **RectangleWin** angenommen (d.h. alle Werte im Zeitbereich werden gleich gewichtet). Mit der Funktion **ConvToSpectrum-Trace** kann aus dem erzeugten Array eine Spektrum-Kurve gebildet werden, welche danach in ein Spektrum-Fenster (FFT Kurvenanzeige) gezogen werden kann.
	
	Example:
	`tr = c0A1;`
	`arr1 = FFT(tr)`
	`arr2 = FFT(tr, HammingWin)`

	`trSpec1 = ConvToSpectrumTrace(arr1)` `trSpec2 = ConvToSpectrumTrace(arr2)`

FlatTopWin	Schlüsselwort für den Parameter **window** der Funktion **FFT**, definiert das **Bewertungsfenster** beim Ausführen der Fast Fourier Transformation.
	
	Example:
	`sineTrace = Sinus(1, 1e3, 0, 1e6, 1e4)`
	`fftVal = FFT(sineTrace, FlatTopWin)`

FreqAtMax(spectrum)	Bestimmt die Frequenz mit der höchsten Amplitude. Die Kurve **spectrum** ist ein Amplitudenspektrum und muss evtl. erst aus einem Zeitsignal berechnet werden. Rückgabewert in Herz [Hz] ist ein Skalar vom Typ double.
	
	Example:
	`tr = c0A1`
	`arrFFT = fft(tr, HammingWin)`
	`trSpec = ConvToSpectrumTrace(arrFFT)`
	`fAtMax = FreqAtMax(trSpec)`

HammingWin	Schlüsselwort für den Parameter **window** der Funktion **FFT**, definiert das **Bewertungsfenster** beim Ausführen der Fast Fourier Transformation.
	
	Example:
	`sineTrace = Sinus(1, 1e3, 0, 1e6, 1e4)`
	`fftVal = FFT(sineTrace, HammingWin)`

HannWin	Schlüsselwort für den Parameter **window** der Funktion **FFT**, definiert das **Bewertungsfenster** beim Ausführen der Fast Fourier Transformation.
	
	Example:
	`sineTrace = Sinus(1, 1e3, 0, 1e6, 1e4)`
	`fftVal = FFT(sineTrace, HannWin)`

IFFT(a)	Inverse Fourier Transformation. Damit kann ein Array aus komplexen Werten **a** im Frequenzbereich in ein Array im Zeitbereich gewandelt werden. Wenn eine Übereinstimmung mit der Original-Kurve vorausgesetzt wird muss die vorgängige **FFT** Berechnung mit der Rechteck-Gewichtung (**RectangleWin**) durchgeführt worden sein.
	
	Example:
	`tr_orig = C0A1`
	
	`; parameters for later calculations`
	`sr = 1/TSample(tr_orig)`
	`Tb =TBegin(tr_orig)`
	
	`; calculate FFT array`
	`arr_fft  = FFT(tr_orig, RectangleWin)`
	
	`; convert back to time array and a trace`
	`arr_time = ifft (arr_fft)`
	`tr_new = ConvToTrace(arr_time, sr)`
	
	`; Adjust x axis`
	`tr_new = Shift(tr_new, Tb)`

Imag(a)	Gibt den/die imaginären Anteile von komplexen Zahlen zurück. **a** kann eine einzelne komplexe Zahl, oder ein Array aus komplexen Zahlen (z.B. Resultat von FFT) sein. Das Resultat wird wieder eine einzelne Zahl oder ein Array aus Double-Zahlen sein. Siehe auch **Abs, Angle, Conjugate, Real**
	
	Example:
	`x = -3+5j ; complex number`
	
	`re = Real(x)  ; re = -3`
	`im = Imag(x)  ; im = 5`
	`ar = angle(x)  ; ar = 120.96°`
	`co = Conjugate(x) ; co = -3-5j`
	
	`; alternatively to  function angle()`
	`deg = atan(imag(x),real(x)) ; deg = 120.96°`

Real(a)	Gibt den/die realen Anteile von komplexen Zahlen zurück. **a** kann eine einzelne komplexe Zahl, oder ein Array aus komplexen Zahlen (z.B. Resultat von FFT) sein. Das Resultat wird wieder eine einzelne Zahl oder ein Array aus Double-Zahlen sein. Siehe auch **Abs, Angle, Conjugate, Imag**
	
	Example:
	`x = -3+5j ; complex number`
	
	`re = Real(x)  ; re = -3`
	`im = Imag(x)  ; im = 5`
	`ar = angle(x)  ; ar = 120.96°`
	`co = Conjugate(x) ; co = -3-5j`
	
	`; alternatively to  function angle()`
	`deg = atan(imag(x),real(x)) ; deg = 120.96°`

RectangleWin	Schlüsselwort für den Parameter **window** der Funktion **FFT**, definiert das **Bewertungsfenster** beim Ausführen der Fast Fourier Transformation.
	
	Example:
	`sineTrace = Sinus(1, 1e3, 0, 1e6, 1e4)`
	`fftVal = FFT(sineTrace, RectangleWin)`

Taylor60Win	Schlüsselwort für den Parameter **window** der Funktion **FFT**, definiert das **Bewertungsfenster** beim Ausführen der Fast Fourier Transformation.
	
	Example:
	`sineTrace = Sinus(1, 1e3, 0, 1e6, 1e4)`
	`fftVal = FFT(sineTrace, Taylor60Win)`

Taylor80Win	Schlüsselwort für den Parameter **window** der Funktion **FFT**, definiert das **Bewertungsfenster** beim Ausführen der Fast Fourier Transformation.
	
	Example:
	`sineTrace = Sinus(1, 1e3, 0, 1e6, 1e4)`
	`fftVal = FFT(sineTrace, Taylor80Win)`

TriangleWin	Schlüsselwort für den Parameter **window** der Funktion **FFT**, definiert das **Bewertungsfenster** beim Ausführen der Fast Fourier Transformation.

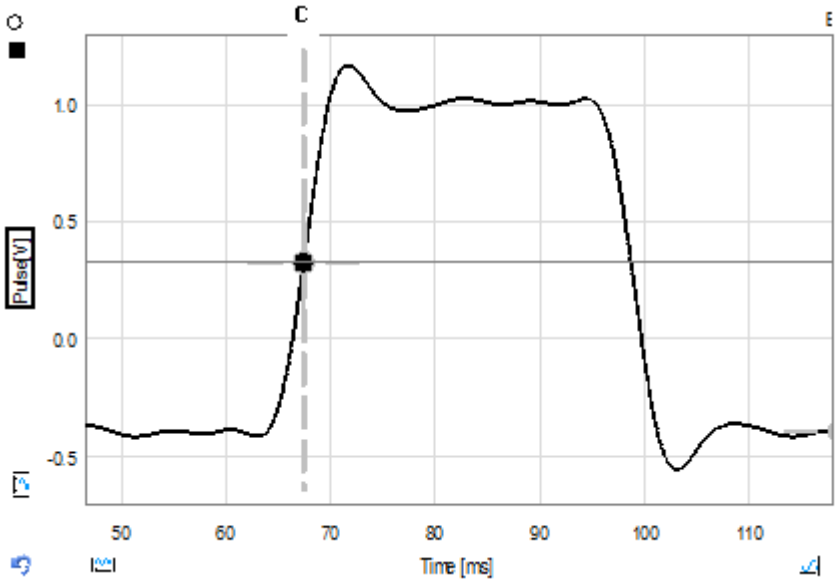
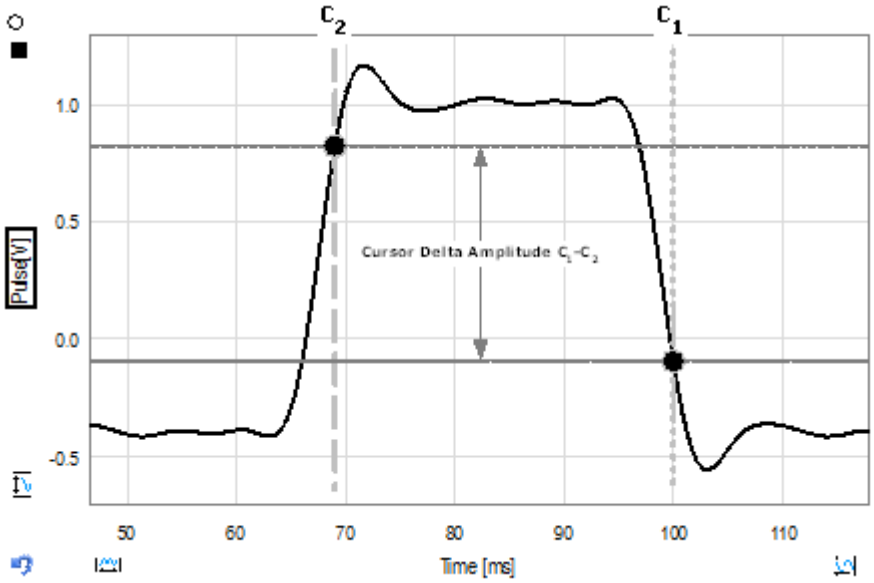
	Example:
	`sineTrace = Sinus(1, 1e3, 0, 1e6, 1e4)`
	`fftVal = FFT(sineTrace, TriangleWin)`

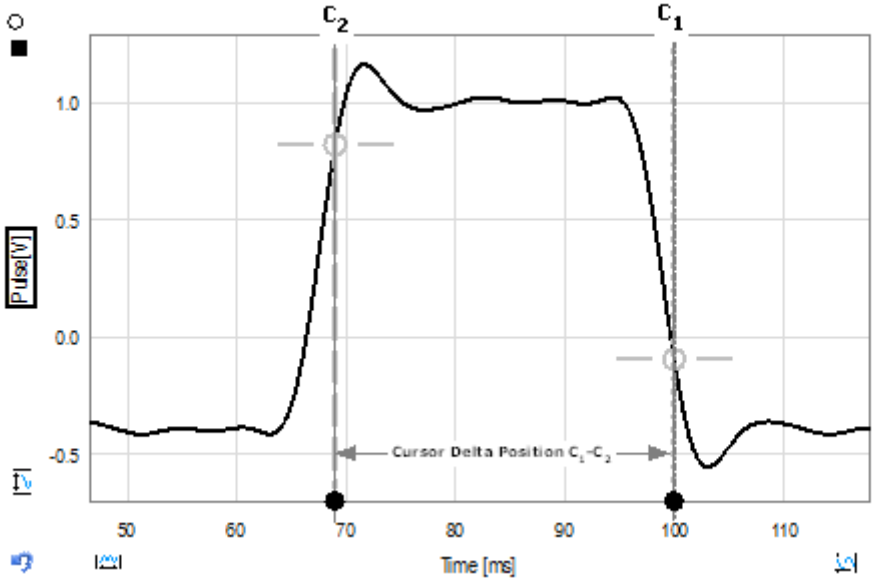
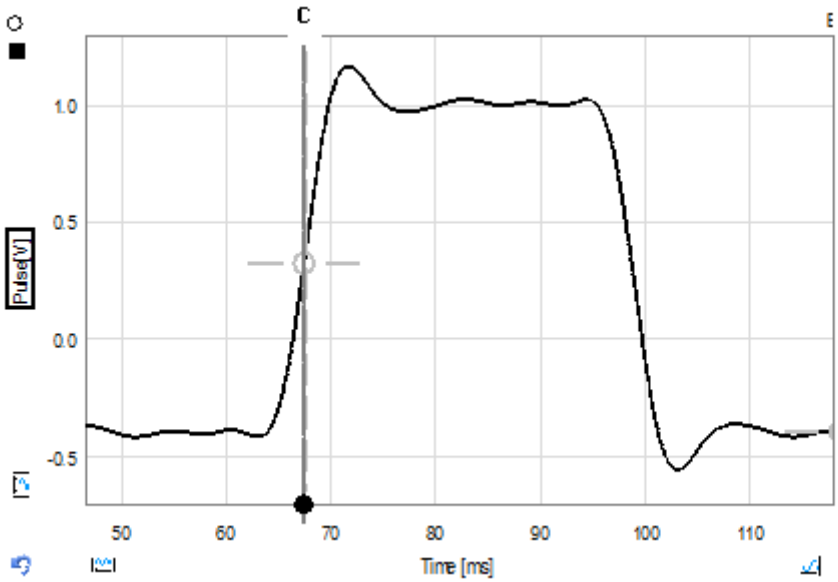
WelchWin	Schlüsselwort für den Parameter **window** der Funktion **FFT**, definiert das **Bewertungsfenster** beim Ausführen der Fast Fourier Transformation.
	
	Example:
	`sineTrace = Sinus(1, 1e3, 0, 1e6, 1e4)`
	`fftVal = FFT(sineTrace, WelchWin)`

35 Liste der Skalar Funktionen

Alle Skalar Funktionen sind auch im Formel Editor als Formelfunktionen verfügbar.

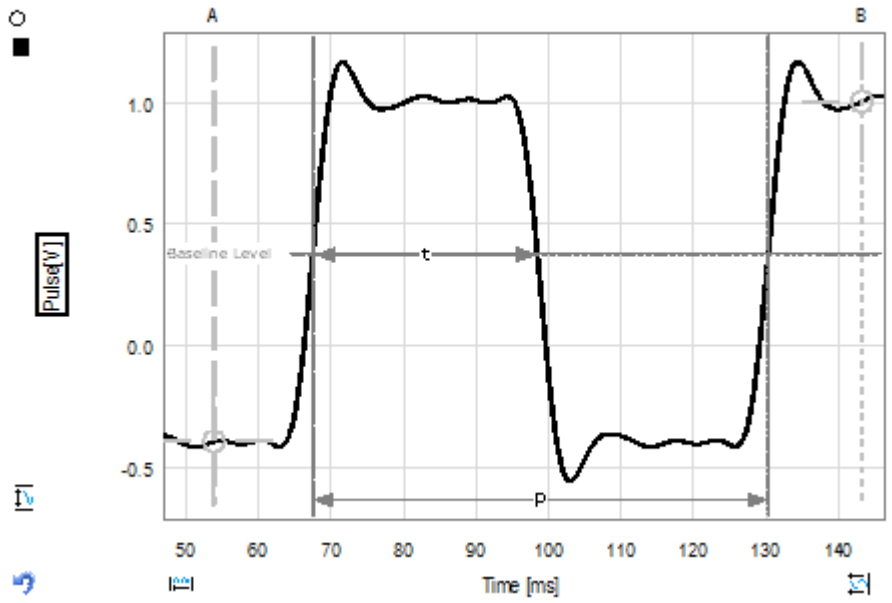
35.1 Group Cursor

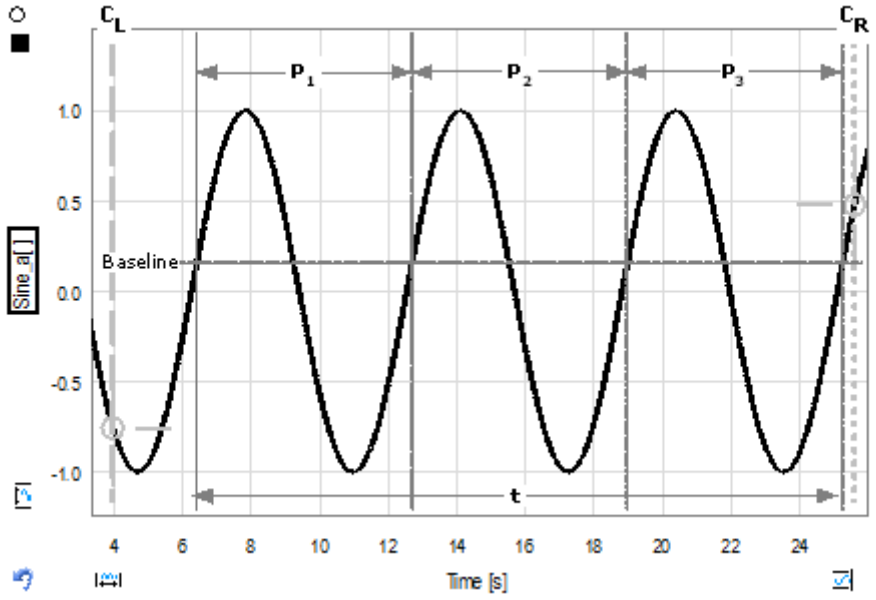
Cursor Amplitude	Gibt den Amplituden-Wert der Kurve and der Position des selektierten Cursors aus.  Cursor Amplitude $= y_C$
Cursor Delta Amplitude	Gibt die Amplituden-Differenz der beiden selektierten Cursoren aus.  Cursor Delta Amplitude $= y_{C1} - y_{C2}$
Cursor Delta Position	Gibt die Differenz der beiden Cursor-Positionen zurück.

	 Cursor Delta Position $= x_{C1} - x_{C2}$
Cursor Position	Gibt die Position des selektierten Cursors auf der X-Achse zurück.  Cursor Position $= x_C$
Cursor Ratio Amplitude (dB)	Berechnet das Amplitudenverhältnis zwischen Cursor 1 und Cursor 2 und skaliert dieses in dB. Diese Skalarfunktion ist nur für Spektren (FFT) verfügbar Für **Peak, RMS und Phase** (Einstellung der Y-Skala) gilt: $CursorRation[dB] = 20 * log_{10}(\frac{Cursor_1}{Cursor_2})$

	Für **Power und RMS^2** (Einstellung der Y-Skala) gilt: $CursorRation[dB] = 10 * log_{10}(\frac{Cursor_1}{Cursor_2})$

35.2 Group Horizontal

Duty Cycle	Duty Cycle gibt das Verhältnis zwischen positiver Pulsbreite und Perioden-Dauer beim Basislinien-Pegel an.  Duty Cycle $= \frac{t}{P}$ t: Pulsbreite bei Basislinien-Pegel P: Periode bei Basislinien-Pegel
Frequency	Sucht die Basislinie-Durchgänge des Signals und berechnet daraus die mittlere Frequenz. Die Durchgänge werden mit Hilfe der eingestellten Hysterese gegenüber Rauschen unempfindlicher gemacht.

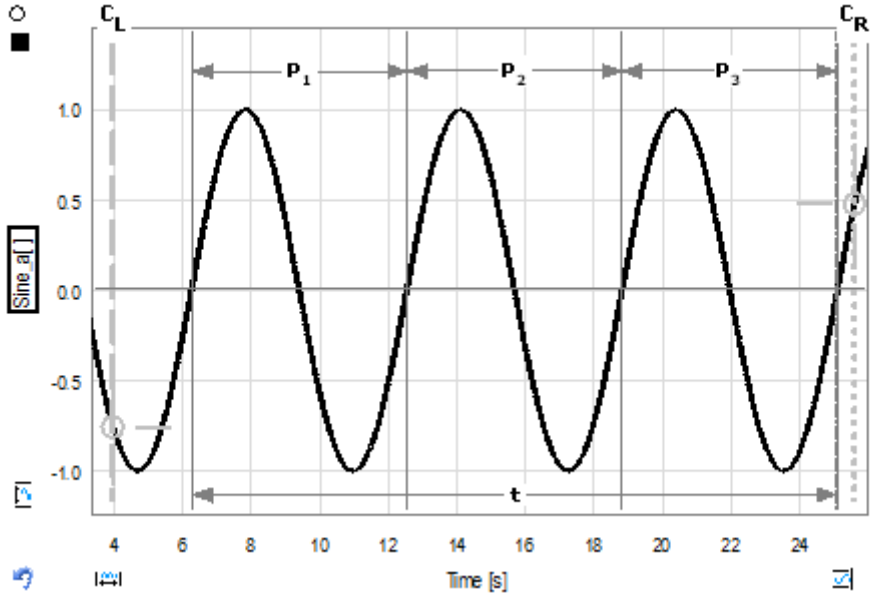


$$Frequency = \frac{N}{t}$$

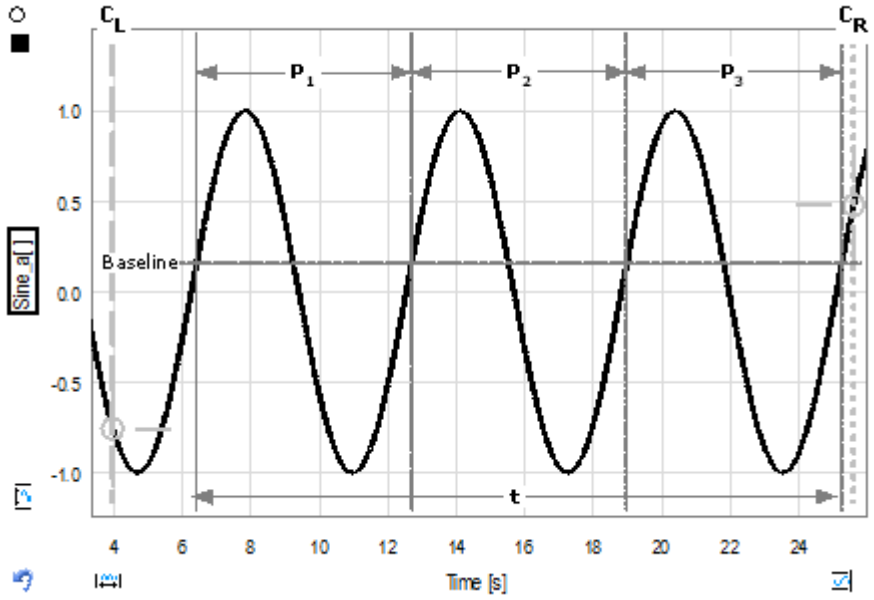
C_L: Cursor links
C_R: Cursor rechts
P_x: Periode
N: Anzahl Perioden
t: Zeit zwischen erstem und letztem Basislinien-Pegel Crossing

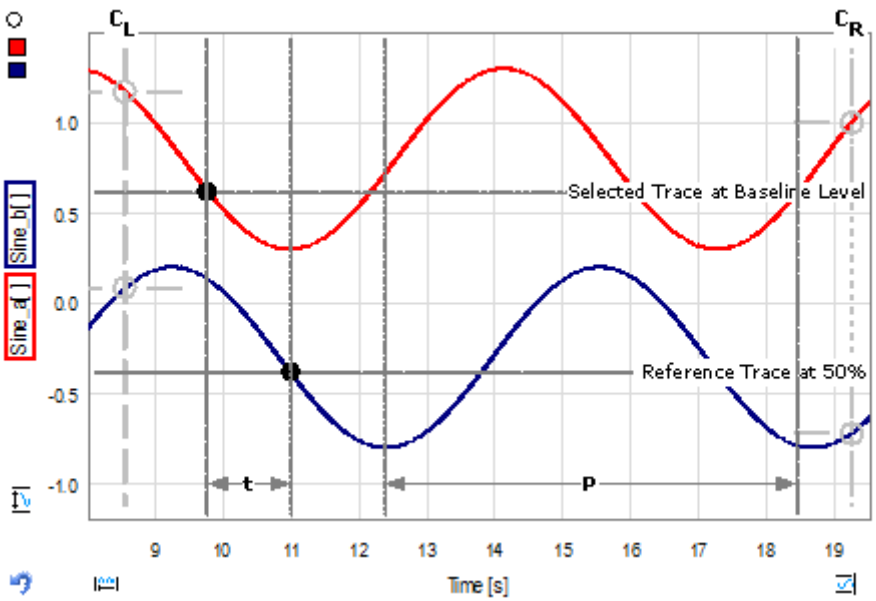
Number of Periods

Sucht die Basislinie-Durchgänge des Signals und zählt die Perioden. Die Durchgänge werden mit Hilfe der eingegebenen Hysterese gegen Rauschen unempfindlicher gemacht.



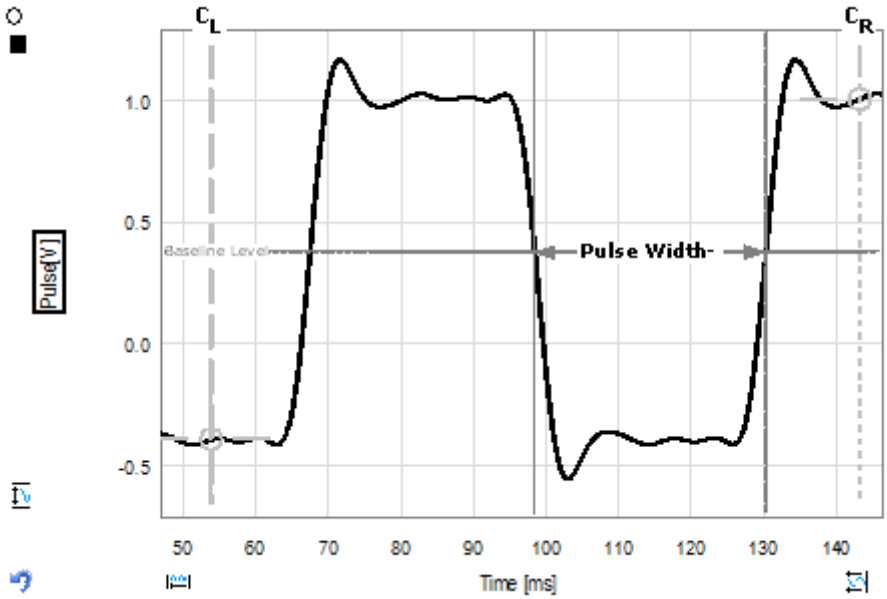
$$\text{Number of Periods} = \frac{t}{P}$$

	C_L: Cursor links C_R: Cursor rechts P_x: Periode t: Zeit zwischen erstem und letztem Durchgang
Period	Sucht die Basislinie-Durchgänge des Signals und berechnet daraus die mittlere Periodendauer. Die Durchgänge werden mit Hilfe der eingegebenen Hysterese gegen Rauschen unempfindlicher gemacht.  $Period = \frac{t}{N}$ C_L: Cursor links C_R: Cursor rechts N: Anzahl Perioden t: Zeit zwischen erstem und letztem Crossing
Phase	Berechnet die Phasenlage von zwei periodischen Signalen mit gleicher Frequenz. Das Referenz-Signal kann aus der Dropdown-Liste ausgewählt werden und wird bei 50% des Amplitudenwertes ausgewertet. Die Berechnung erfordert mindestens 3.5 Perioden. Ein negatives Resultat wird zurückgegeben, wenn das Referenz-Signal nacheilt. Der Wertebereich der Phase ist +/-180°.

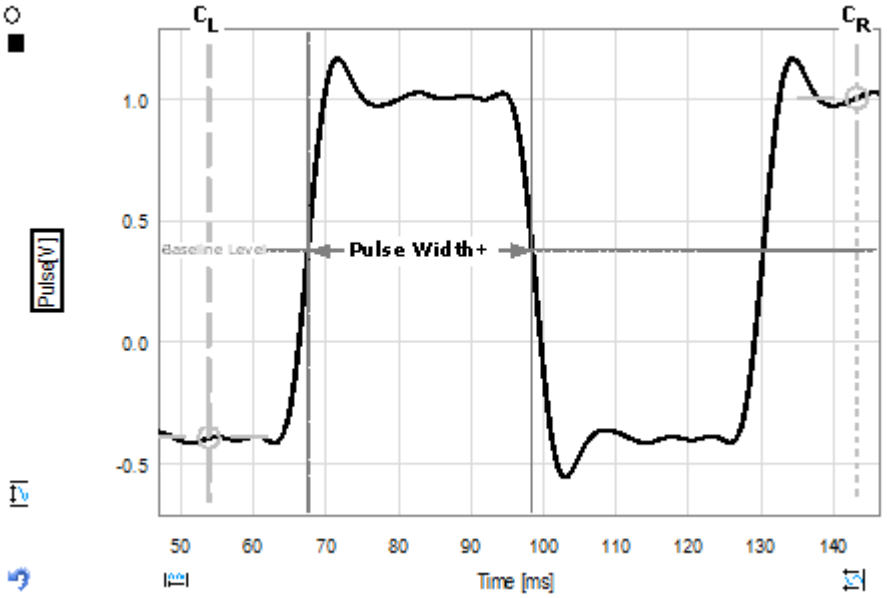
<table>
<tr>
<td></td>
<td>


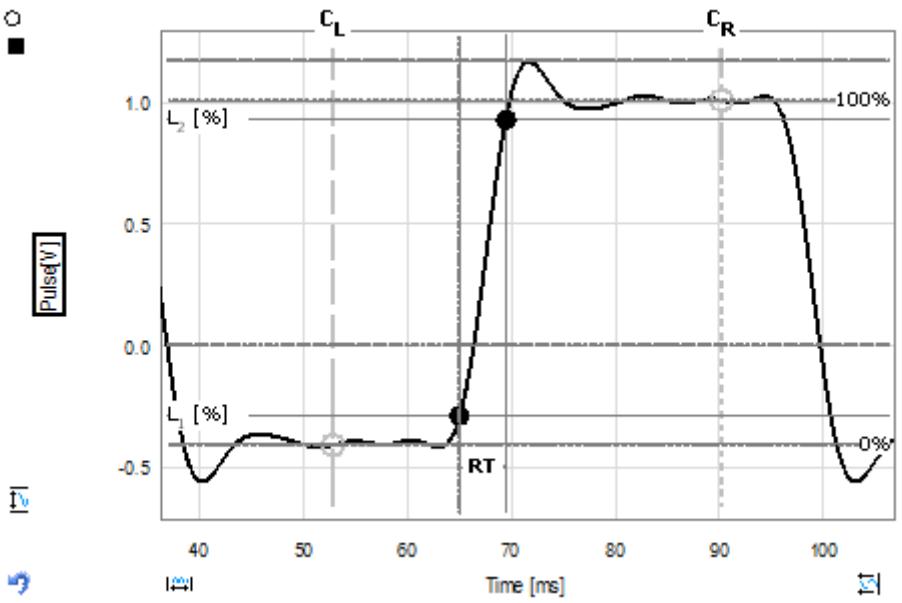
$$Phase = 360 \cdot \frac{t}{P}$$

C_L: Cursor links
C_R: Cursor rechts
P: Periode
t: Zeitdifferenz von Basislinie bis 50% der Amplitude
</td>
</tr>
<tr>
<td>Pulse Width (neg)</td>
<td>
Berechnet die negative Pulsbreite beim Pegel der Basislinie. Falls mehrere Perioden zwischen den Cursoren liegen, wird die mittlere Pulsbreite berechnet.



C_L: Cursor links
C_R: Cursor rechts
</td>
</tr>
</table>

	Pulse Width-: Negative Pulsbreite beim Pegel der Basislinie
Pulse Width (pos)	Berechnet die positive Pulsbreite beim Pegel der Basislinie. Falls mehrere Perioden zwischen den Cursoren liegen, wird die mittlere Pulsbreite berechnet.  C_L: Cursor links C_R: Cursor rechts Pulse Width+: Positive Pulsbreite beim Pegel der Basislinie
Rise/Fall time	Ermittelt die Anstiegs- oder Abfallzeit einer Kurve bei gewünschten Pegeln [%]. • Um die Anstiegszeit zu berechnen soll der linke Cursor auf die Grundlinie (0%) und der rechte Cursor auf die Schulter (100%) des Pulses positioniert werden. • Um die Abfallzeit zu berechnen soll der linke Cursor auf die Schulter (100%) und der rechte Cursor auf die Grundlinie (0%) des Pulses positioniert werden. Für diese Skalarfunktion müssen zwei Pegel eingestellt werden (z.B. 10% und 90%). Diese bestimmen die Punkte auf der Kurvenflanke zur Ermittlung der Anstiegs- bzw. Abfallzeit.



C_L: Cursor links (0% Amplitude)

C_R: Cursor rechts (100% Amplitude)

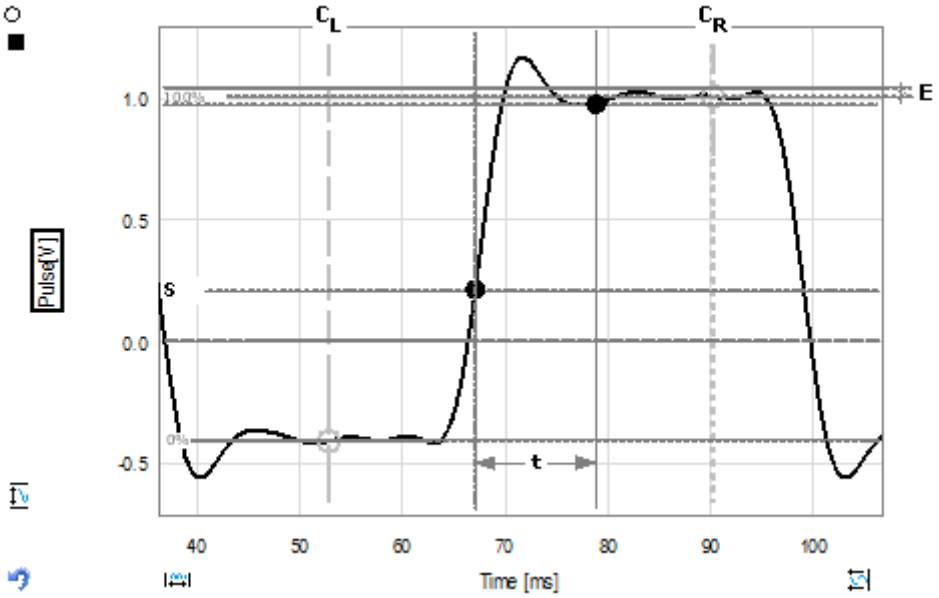
$L_1[\%]$: Level 1 (Unterer Level [%] für Anstiegszeit)

$L_2[\%]$: Level 2 (Oberer Level [%] für Anstiegszeit)

RT: Anstiegszeit

Settling time

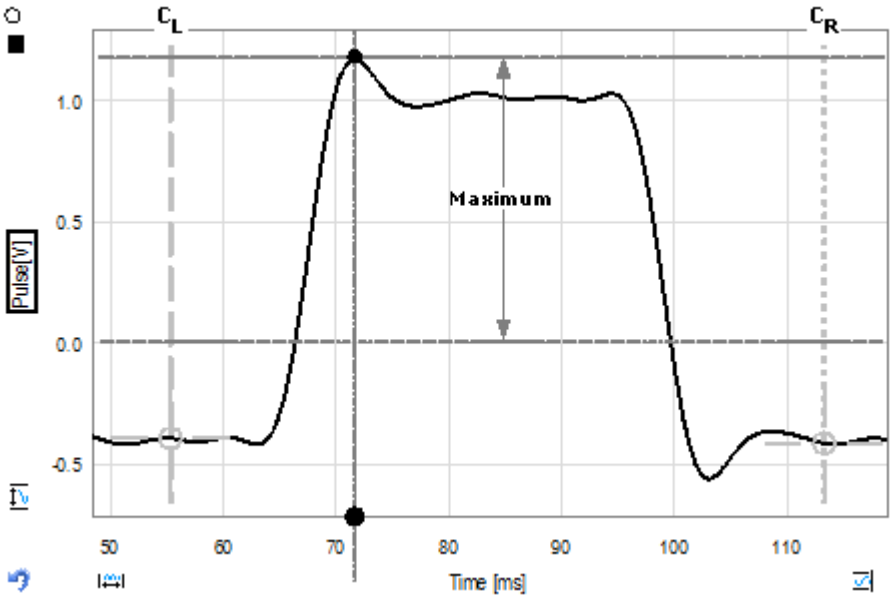
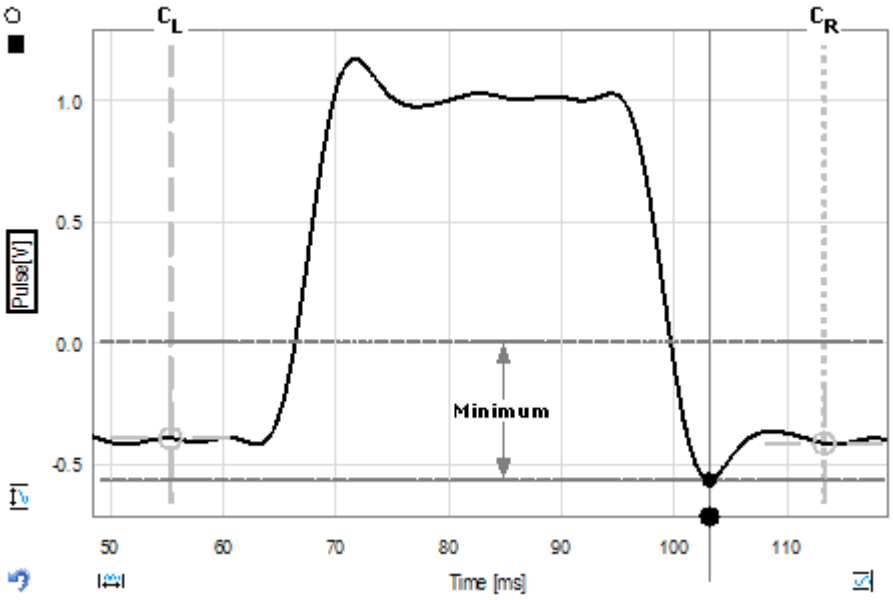
Gibt die Zeit zurück bis das Signal innerhalb eines Fehlerbandes bleibt. Das Fehlerband bezieht sich auf den Endwert (100%), der festgelegt wird durch den rechten Cursor.



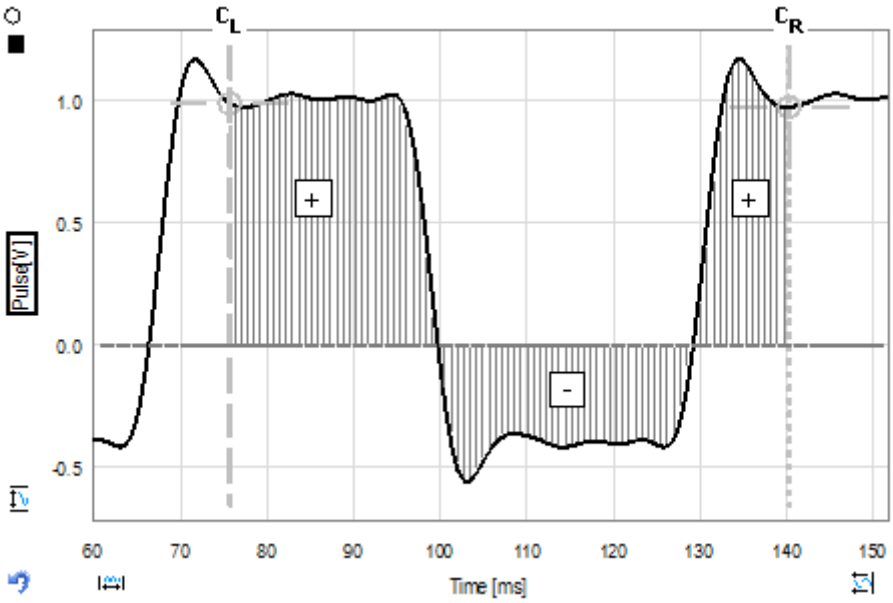
C_L: Cursor links (0%)

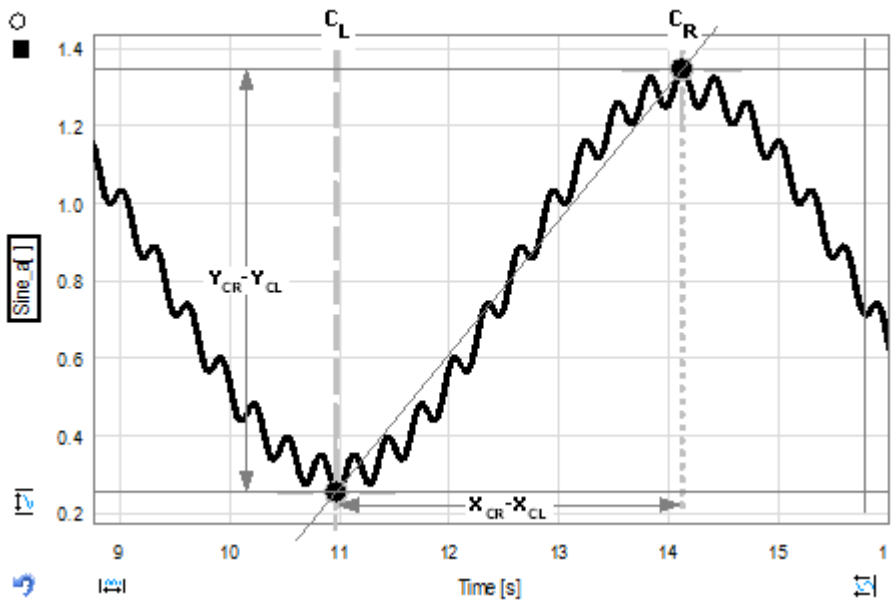
C_R: Cursor rechts (100%)

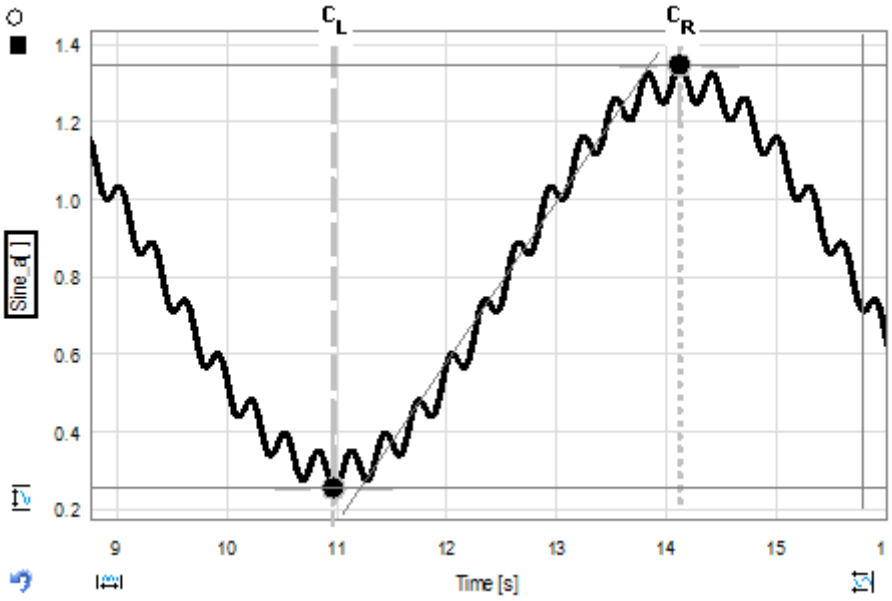
S: Start-Level [%]

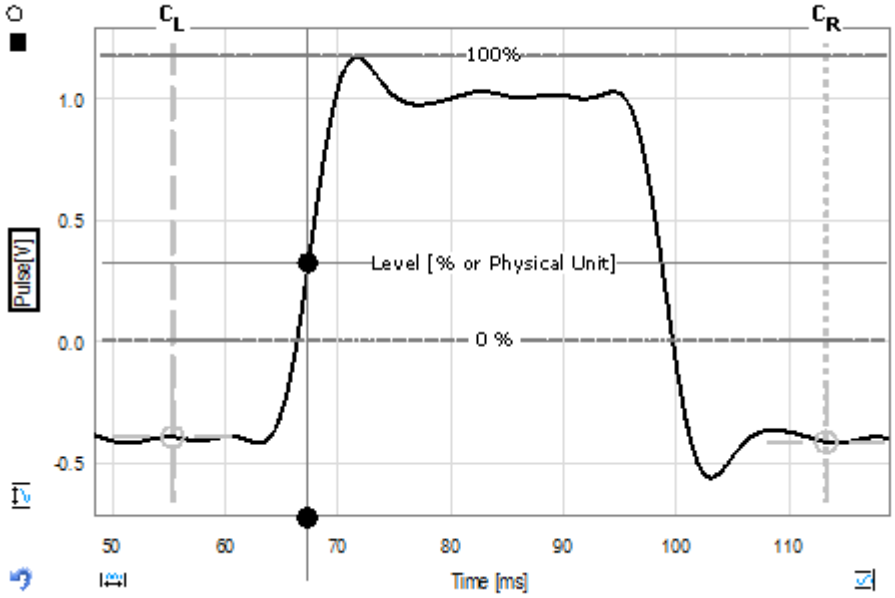
	E: End-Level - Halbes Fehlerband [%] t: Settling time
Time at Maximum	Sucht das Maximum und gibt die Zeit an der Position des Maximums zurück.  C_L: Cursor links C_R: Cursor rechts
Time at Minimum	Sucht das Minimum und gibt die Zeit an der Position des Minimums zurück.  C_L: Cursor links C_R: Cursor rechts

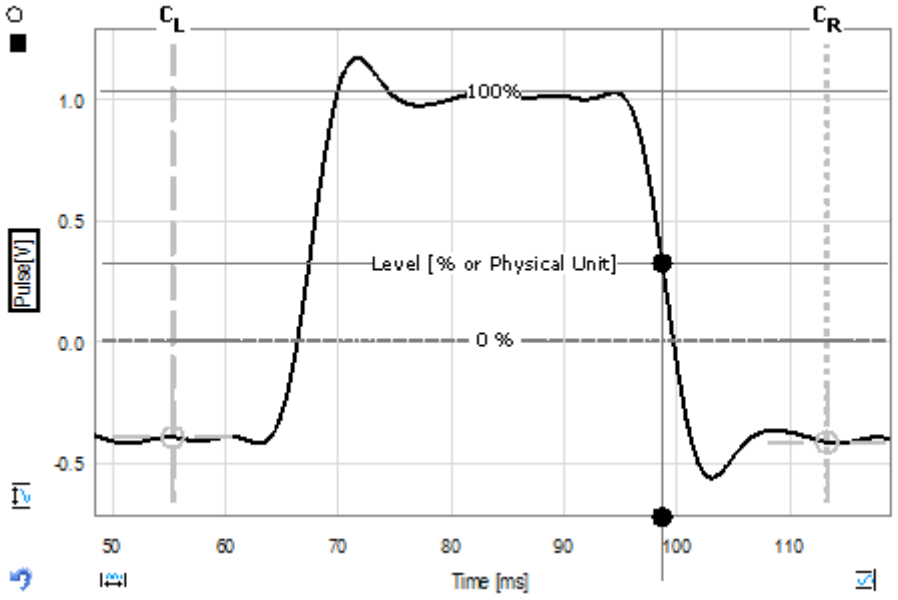
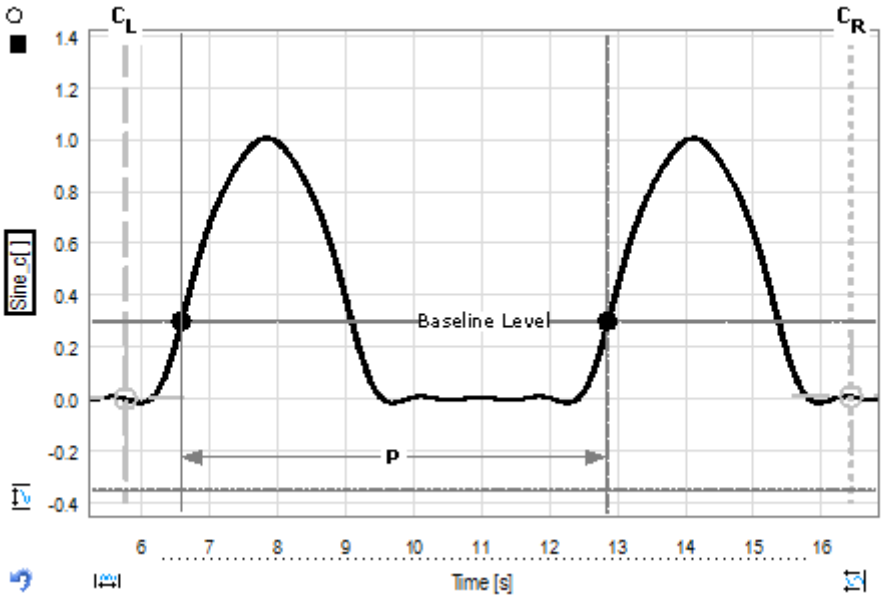
35.3 Group Misc

Area	Berechnet die Fläche unter der Kurve zwischen den Cursoren relativ zum Nullpunkt. Werte oberhalb der Nulllinie werden als positive Flächenwerte berechnet, Werte unterhalb der Nulllinie als negative Flächenwerte.  $Area = \sum_{i=C_L}^{C_R} y_i \cdot \Delta t$ C_L: Cursor links C_R: Cursor rechts y_i: Y-Wert an Position i Δt: Abtastintervall
Cadence	Berechnet die Frequenz normiert auf die Minute. $Cadence = Frequency \cdot 60$ $Frequency$: Berechneter Skalarwert
Energy	Berechnet das Integral der quadrierten Punkte. $Energy = \sum_{i=C_L}^{C_R} y_i^2 \cdot \Delta t$ C_L: Cursor links C_R: Cursor rechts y_i: Y-Wert an Position i Δt: Abtastintervall
FormulaVariable	Fügt einen berechneten Skalarwert aus einer TranAX Formel in die Skalartabelle ein.

	Nur für Skalar Tabelle B sinnvoll verwendbar
Number of Blocks	Gibt die totale Anzahl Blöcke an während die Erfassung läuft. Dieser Wert ist unabhängig von den Cursoren. Number of Blocks wird gebraucht in den Block- und ECR Datenerfassungsmodi.
Number of Triggers	Gibt die totale Anzahl Trigger-Ereignisse an. Dieser Wert ist unabhängig von den Cursroren. Number of Triggers wird gebraucht in den Block- und ECR Datenerfassungsmodi.
Overflow	Overflow beobachted den positiven Überlauf des ADC's des Eingangsverstärkers. • Gibt 'No' zurück in der Hintergrundfarbe Grün falls kein Überlauf festgestellt wird. • Gibt 'Yes' zurück in der Hintergrundfarbe Rot und der Angabe des Zeitpunktes des ersten Überlaufes.
Slope	Berechnet die Steigung zwischen dem Schnittpunkt der Kurve und den Cursoren. 

	$Slope = \frac{y_{CR} - y_{CL}}{x_{CR} - x_{CL}}$ y_{CL}: Y-Wert an Position des linken Cursors y_{CR}: Y-Wert an Position des rechten Cursors x_{CL}: X-Wert an Position des linken Cursors x_{CR}: X-Wert an Position des rechten Cursors
Slope (Lin. Regr.)	Berechnet die Steigung der Kurve mittels linearer Regression zwischen den Cursoren.  Slope by Linear Regression C_L: Cursor links C_R: Cursor rechts
Text	Das ist ein simpler Platzhalter. Er ermöglicht einen beliebigen Text in der Tabelle einzutragen; zwecks Reports oder zur Strukturierung der Tabelle. Diese Funktion ist nur bei Skalarfunktionen Tabelle B vorhanden
Trigger Real Time	Gibt die Zeit des Triggerereignisses in relativer oder absoluter Zeit zurück. In den Betriebsarten Multiblock oder ECR wird das Resultat aus dem nächstgelegenen Datenblock ermittelt zwischen den angewählten Cursor. Der Scope- und Continuous Mode liefern nur sinnvolle Resultate in absoluter Zeit.
TxLeft	Gibt die Zeit zurück, bei der in einem positiven Puls ein zu definierender Pegel ansteigend überschritten wird. In diesem Fall wird vom linken Cursor ausgegangen. Der Pegel kann in % vom Maximum (Nulllinie entspricht 0%, Maximum entspricht 100%) oder als physikalische Grösse angegeben werden. Falls Abs. Max angewählt ist, so wird zuerst die absolute Kurve berechnet

	bevor TxLeft ermittelt wird. Der Nullpunkt hängt von den Einstellungen der X-Achse ab wie Relative Zeit oder Relative Zeit mit Nullpunkt beim Trigger.  C_L: Cursor links C_R: Cursor rechts
TxRight	Gibt die Zeit zurück, bei der in einem positiven Puls ein zu definierender Pegel ansteigend überschritten wird. In diesem Fall wird vom rechten Cursor ausgegangen. Der Pegel kann in % vom Maximum (Nulllinie entspricht 0%, Maximum entspricht 100%) oder als physikalische Grösse angegeben werden. Falls Abs. Max angewählt ist, so wird zuerst die absolute Kurve berechnet bevor TxLeft ermittelt wird. Der Nullpunkt hängt von den Einstellungen der X-Achse ab wie Relative Zeit oder Relative Zeit mit Nullpunkt beim Trigger.

	 C_L: Cursor links C_R: Cursor rechts
UnderFlow	UnderFlow beobachted den negativen Überlauf des ADC's des Eingangsverstärkers. • Gibt 'No' zurück in der Hintergrundfarbe Grün falls kein Überlauf festgestellt wird. • Gibt 'Yes' zurück in der Hintergrundfarbe Rot und der Angabe des Zeitpunktes des ersten Überlaufes.
Velocity	Berechnet die Geschwindigkeit aus Periodendauer und vorgegebenem Weg.  $Velocity = \frac{\text{Sensor Distance}}{P}$

	Sensor Distance: Sensor-Distanz [m] P: Periode

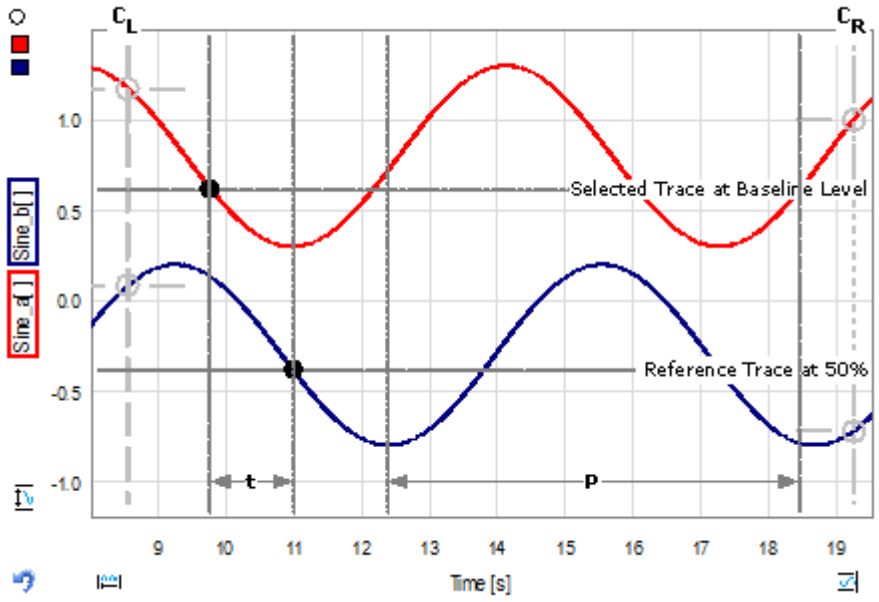
35.4 Group Periodic

Crest Factor Periodic	Der Crest-Faktor beschreibt das Verhältnis zwischen Spitzenwert und RMS von periodischen Signalen. $$CrestFactor_{Periodic} = \frac{\lvert Peak \rvert}{RMS_{Periodic}}$$ $Peak$: Absolute Peak $RMS_{Periodic}$: Periodic Root Mean Square
Mean Periodic	Sucht die Basislinie-Druchgänge des Signals und berechnet den Mittelwert nur für die gefundenen ganzgen Perioden. $$Mean_{Periodic} = \frac{1}{n \cdot \Delta t} \cdot \sum_{i=LC_L}^{LC_R} y_i \cdot \Delta t$$ LC_L: Level crossing links LC_R: Level crossing rechts n: Anzahl Samples y_i: Y-Wert an Position i Δt: Abtastintervall
Rectified Mean Periodic	Sucht die Durchgänge der Basislinie und berechnet den gleichgerichteten Mittelwert des Signals von ganzen Perioden. $$Rect_{Periodic} = \frac{1}{n \cdot \Delta t} \cdot \sum_{i=LC_L}^{LC_R} \lvert y_i \rvert \cdot \Delta t$$ LC_L: Level Crossing links LC_R: Level Crossing rechts n: Anzahl Samples y_i: Y-Wert an Position i Δt: Abtastintervall
RMS Periodic	Sucht die Durchgänge der Basislinie und berechnet den Effektivwert der ganzen Perioden. $$RMS_{Periodic} = \sqrt{\frac{1}{n \cdot \Delta t} \cdot \sum_{i=LC_L}^{LC_R} y_i^2 \cdot \Delta t}$$ LC_L: Level crossing links LC_R: Level crossing rechts n: Anzahl Samples y_i: Y-Wert an Position i Δt: Abtastintervall

StdDev Periodic	Sucht die Durchgänge der Basislinie und berechnet die Standardabweichung des Signals von ganzen Perioden. Die periodische Standardabweichung entspricht einer periodischen RMS-Berechnung mit Abzug des Offsets. $SDev_{Periodic} = \sqrt{\frac{1}{n \cdot \Delta t} \cdot \sum_{i=LC_L}^{LC_R} (y_i - mean)^2 \cdot \Delta t}$ LC_L: Level crossing links LC_R: Level crossing rechts n: Anzahl Samples y_i: Y-Wert an Position i Δt: Abtastintervall

35.5 Group Power

Apparent Power	Berechnet die Scheinleistung von zwei periodischen Signalen aus Spannung und Strom. Empfohlen wird Skalartabelle B für diese Skalarfuntkion $ApparentPower = RMS_{Periodic}(u) \cdot RMS_{Periodic}(i)$ $RMS_{Periodic}$: Siehe Scalar RMS Periodic u: Spannungskurve i: Stromkurve
Cos(phi)	Berechnet den cos($\mathfrak{U}$) an der gewählten Basislinie aus den Grundwellen von zwei periodischen, frequenzgleichen Signalen (normalerweise Spannung und Strom). Das Referenzsignal wird bei 50% der Amplitude ausgewertet und kann vom entsprechenden Skalarmenü ausgewählt werden. Die Berechnung erfordert mind. 3.5 ganze Perioden. Beim Einsatz von Spannungssonden und Stromzangen können Zeitverschiebungen auftreten verursacht durch verschiedene Bandbreiten und Kabellängen. Solche Phasenverschiebungen sollten kompensiert werden bevor cos(phi) berechnet wird. Die Formel-Funktion Shift kann beigezoden werden, um eine Kurve gegen die andere zeitlich zu verschieben. $cos(\phi) = cos(360 \cdot \frac{T}{P})$

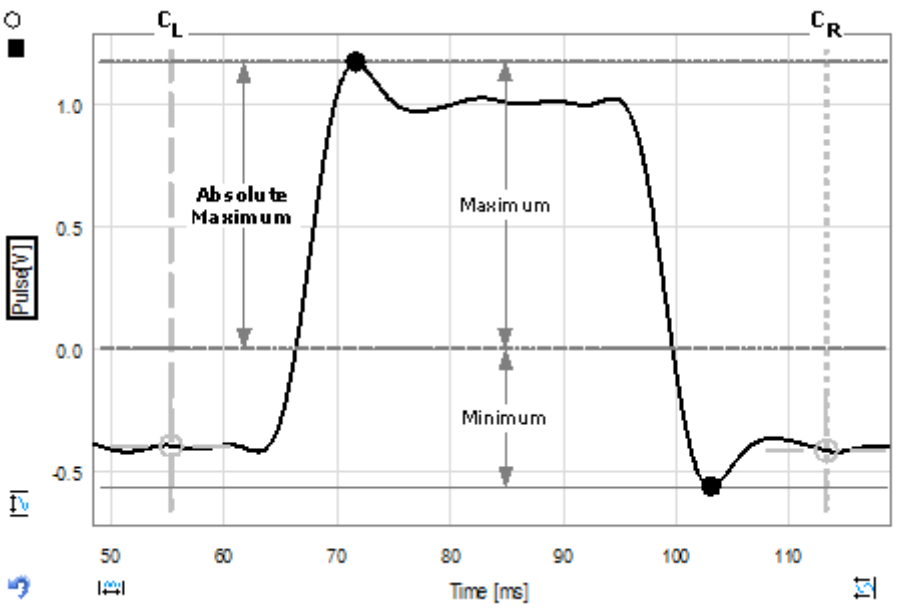
	P: Periode T: Zeitdifferenz zwischen den steigenden Flanken
Crest Factor	Der Crest-Faktor beschreibt das Verhältnis zwischen Spitzenwert und RMS. $CrestFactor = \frac{\lvert Peak \rvert}{RMS}$ $Peak$: Absolute Peak RMS : Root Mean Square
Crest Factor Periodic	Der Crest-Faktor beschreibt das Verhältnis zwischen Spitzenwert und RMS von periodischen Signalen. $CrestFactor_{Periodic} = \frac{\lvert Peak \rvert}{RMS_{Periodic}}$ $Peak$: Absolute Peak $RMS_{Periodic}$: Periodic Root Mean Square
Fundamen-tal Power	Sucht die Basislinie-Durchgänge der Spannungs- und der Stromkurve, um die Periodendauer zu bestimmen und berechnet anschliessend daraus Funda-mental Power, die Wirkleistung in der Grundwelle ohne die Oberwellen. Es wird davon ausgegangen, dass die beiden Signale periodisch sind und dass es sich dabei um ein Spannung- und ein Stromsignal handelt.
Phase	Berechnet die Phasenlage von zwei periodischen Signalen mit gleicher Fre-quenz. Das Referenz-Signal kann aus der Dropdown-Liste ausgewählt wer-den und wird bei 50% des Amplitudenwertes ausgewertet. Die Berechnung erfordert mindestens 3.5 Perioden. Ein negatives Resultat wird zurückgegeben, wenn das Referenz-Signal nacheilt. Der Wertebereich der Phase ist +/-180°. 

	$Phase = 360 \cdot \frac{t}{P}$ C_L: Cursor links C_R: Cursor rechts P: Periode t: Zeitdifferenz von Basislinie bis 50% der Amplitude
Power Factor	Berechnet den Leistungs-Faktor aus zwei periodischen Signalen mit gleicher Frequenz (normalerweise Spannung und Strom). Das Referenz-Signal kann aus der Dropdown-Liste ausgewählt werden. Für diese Funktion sollte bevorzugt Skalartabelle B verwendet werden. Die Berechnung erfordert mind. 1.5 ganze Perioden. Zwischen Spannungs. und Stromkurve können Zeitverschiebungen auftreten, die von unterschiedlichen Bandbreiten und Kabellängen der Tastköpfe herrühren. Eine solcher Zeitversatz entspricht einem Phasenfehler und muss korrigiert werden bevor diese Berechnung durchgeführt wird. Der Formeleditor bietet eine Funktion Shift an, die zur zeitlichen Verschiebung einer Kurve beigezogen werden kann. Power Factor $= \frac{P}{S}$ P: Wirkleistung S: Scheinleistung
Reactive Power	Berechnet die Blindleistung aus Scheinleistung und Wirkleistung. Für diese Funktion sollte bevorzugt Skalartabelle B verwendet werden. $Preact = RMS_{Periodic}(u) \cdot RMS_{Periodic}(i) \cdot sin(\Delta\varphi)$ $RMS_{Periodic}$: Siehe Scalar RMS Periodic u: Spannung i: Strom $\Delta\varphi$: Phasendifferenz zwischen Spannung und Strom
Real Power	Berechnet die Wirkleistung vom Mittelwert der Momentanleistung. Für diese Funktion sollte bevorzugt Skalartabelle B verwendet werden.

	$Preal = RMS_{Periodic}(u) \cdot RMS_{Periodic}(i) \cdot cos(\Delta\varphi)$ $RMS_{Periodic}$: Siehe Scalar RMS Periodic u: Spannung i: Strom $\Delta\varphi$: Phasendifferenz zwischen Spannung und Strom
Total Harmonic Distortion	Sucht zuerst die Grundfrequenz und berechnet dann die Total Harmonic Distorsion (THD) in % eines periodischen Signales. Zur Berecjnung werden mindestens 2 Perioden zwischen dem Cursor-Paar benötigt. Verrauschte Signale müssen eventuell vor der Berechnung mittels einem Tiefpassfilter verbessert werden. Auch sollten die Basislinie und Hysterese sorgfältig gewählt werden. $THD_U = \frac{\sqrt{U_2^2 + U_3^2 + U_4^2 + ... + U_n^2}}{U_1}$ [%] $THD_I = \frac{\sqrt{I_2^2 + I_3^2 + I_4^2 + ... + I_n^2}}{I_1}$ [%] $n = minOf\left(50, \frac{samplingFrequency}{2 \cdot baseFrequency} - 1\right)$ U_1: Spannung bei der Grundfrequenz I_1: Strom bei der Grundfrequenz U_x: x_{th} harmonische von Spannungskurve I_x: x_{th} harmonische von Stromkurve

35.6 Group Vertical

Abs.Max	Sucht den grössten absoluten Kurvenwert. Die Funktion liefert den kleinsten (negativen) Wert, wenn dessen Betrag grösser ist als der grösste positive Wert.



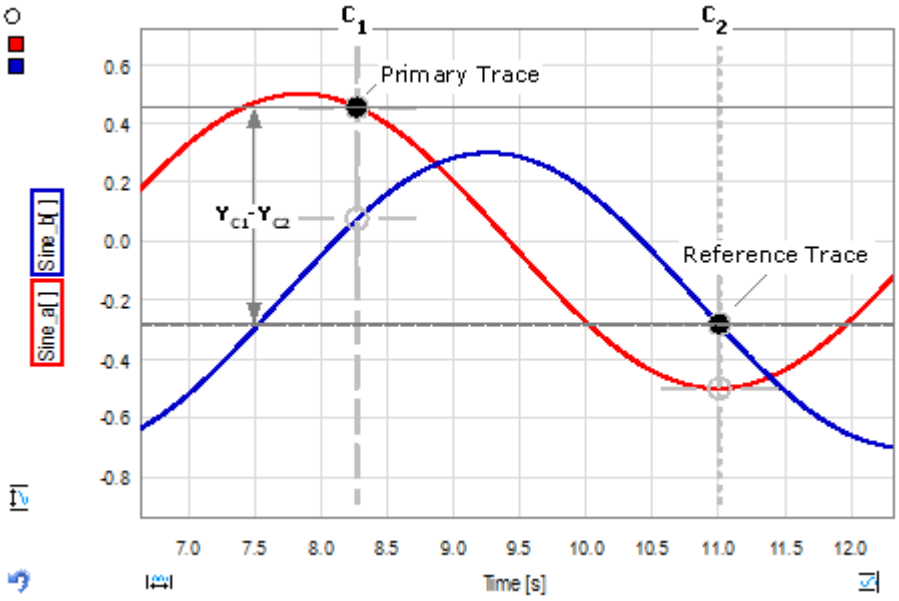
$$Abs.Max = \begin{cases} |Maximum| \ if |Maximum| \geq |Minimum| \\ |Minimum| \ if |Minimum| > |Maximum| \end{cases}$$

C_L: Cursor links
C_R: Cursor rechts

Delta

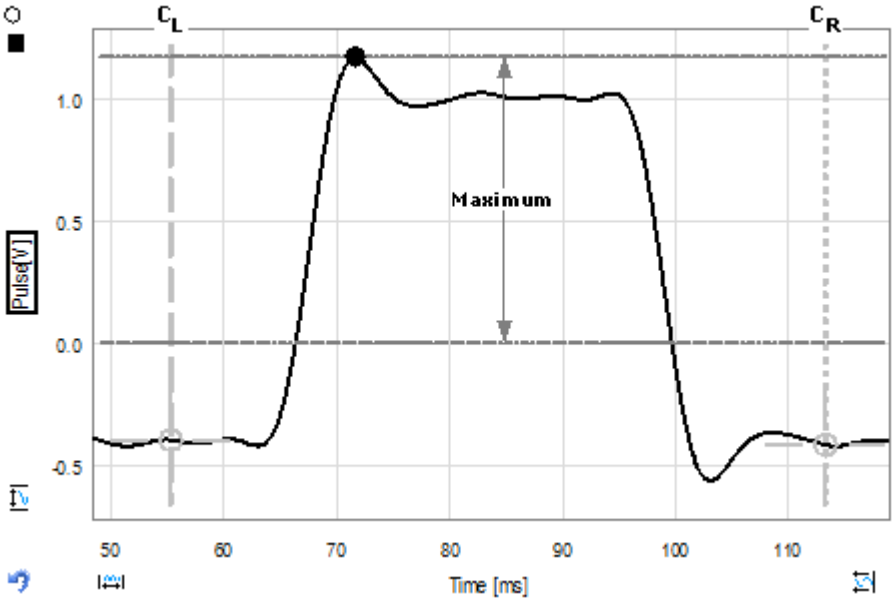
Ermittelt die Amplituden-Differenz zweier Kurven zwischen zwei Cursoren.

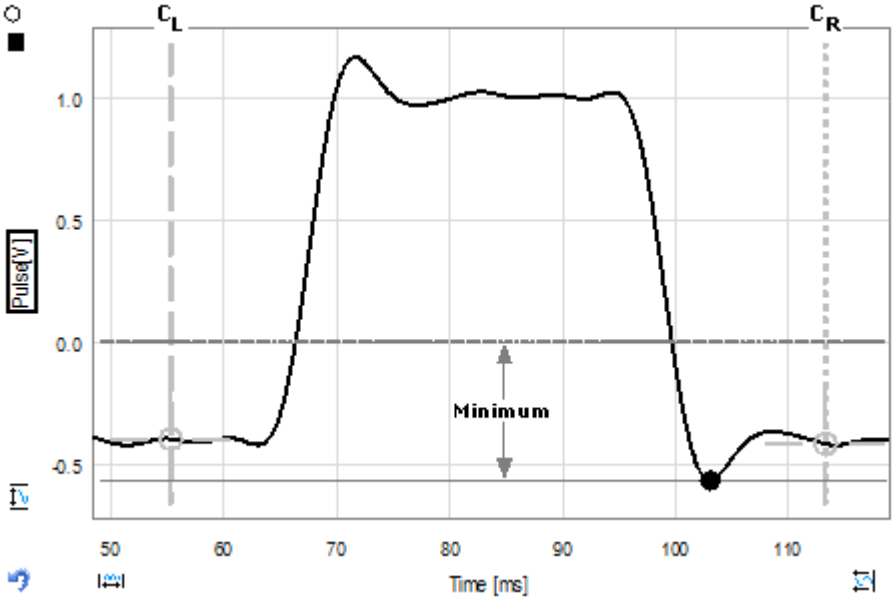
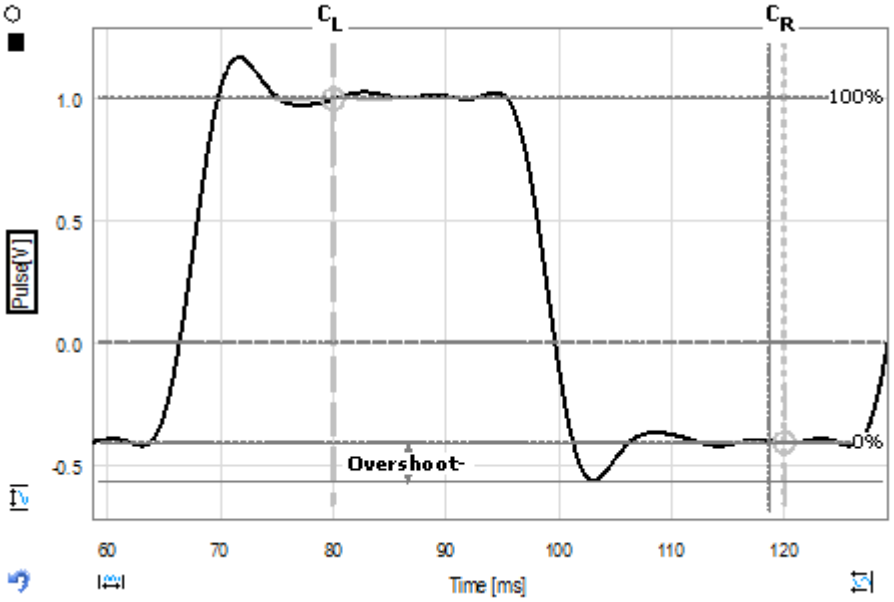
> Die Delta-Funktion erlaubt die Kombination von beliebigen Cursor-Paaren mit zwei unabhängigen Kurven.

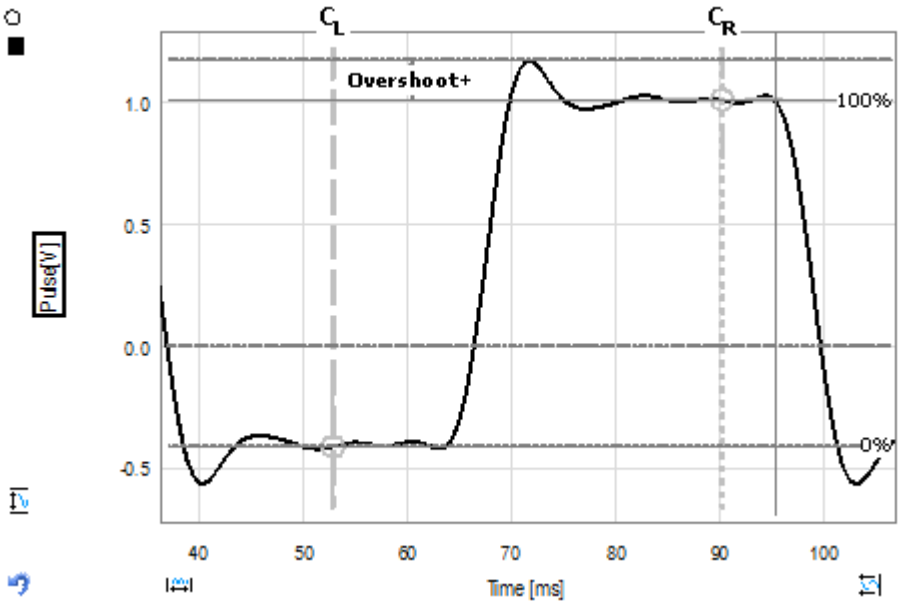


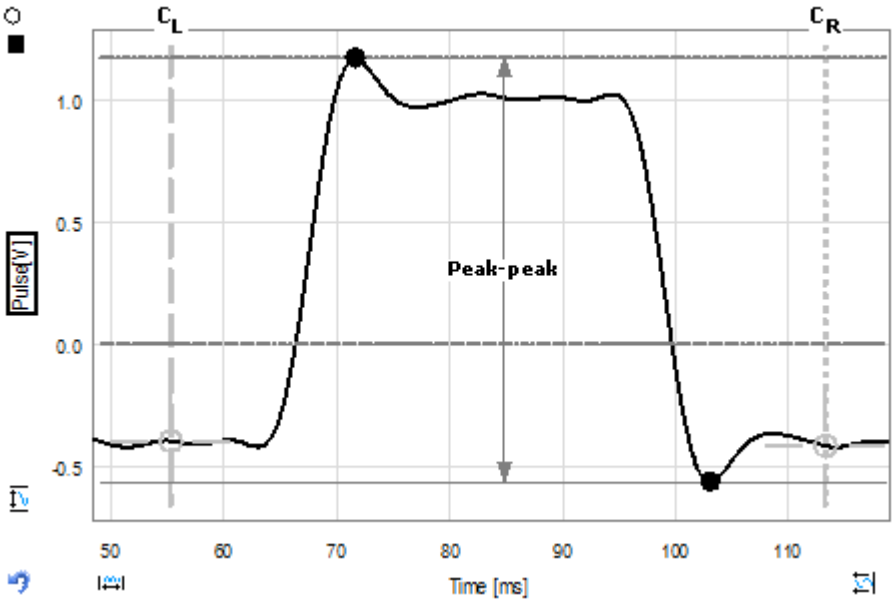
$$Delta = y_{\text{C1 Primary Trace}} - y_{\text{C2 Reference Trace}}$$

C_1: Selektierter Cursor 1

	C_2: Selektierter Cursor 2
Frequency at Maximum (FFT)	Sucht das Maximum zwischen Cursor A und Cursor B und gibt die Frequenz an der Position des Maximums zurück. Diese Skalarfunktion ist nur für Spektren (FFT) verfügbar
Maximum	Sucht den grössten Punkt in der Kurve relativ zur Signal-Nulllinie.  C_L: Cursor links C_R: Cursor rechts
Mean	Berechnet den Mittelwert des Signals. $Mean = \frac{1}{n \cdot \Delta t} \cdot \sum_{i=C_L}^{C_R} y_i \cdot \Delta t$ C_L: Cursor links C_R: Cursor rechts n: Anzahl Samples y_i: Y-Wert an Position i Δt: Abtastintervall
Mean Periodic	Sucht die Basislinie-Druchgänge des Signals und berechnet den Mittelwert nur für die gefundenen ganzgen Perioden. $Mean_{Periodic} = \frac{1}{n \cdot \Delta t} \cdot \sum_{i=LC_L}^{LC_R} y_i \cdot \Delta t$ LC_L: Level crossing links LC_R: Level crossing rechts

	n: Anzahl Samples y_i: Y-Wert an Position i Δt: Abtastintervall
Minimum	Sucht den kleinsten Punkt in der Kurve relativ zur Signal-Nulllinie.  C_L: Cursor links C_R: Cursor rechts
Overshoot-	Gibt die Stärke der Unterschwingung zurück. Der linke Cursor soll auf die Puls-Schulter gelegt werden (100%) und der rechte Cursor auf die Puls-Grundlinie (0%). 

	C_L: Cursor links (Puls-Schulter 100%) C_R: Cursor rechts (Puls-Grundlinie 0%) Overshoot-: Unterschwingung [%]
Overshoot+	Gibt die Stärke der Überschwingung zurück. Der linke Cursor soll auf die Puls-Grundlinie (0%) und der rechte Cursor auf die Puls-Schulter (100%) gelegt werden.  C_L: Cursor links (Puls-Grundlinie 0%) C_R: Cursor right (Puls-Schulter 100%) Overshoot+: Überschwingung [%]
Peak-Peak	Berechnet die Differenz von Maximum und Minimum.

	 Peak-Peak = Maximum − Minimum C_L: Cursor links C_R: Cursor rechts
Rectified Mean	Berechnet den gleichgerichteten Mittelwert des Signals. $Rect = \frac{1}{n \cdot \Delta t} \cdot \sum_{i=C_L}^{C_R} \lvert y_i \rvert \cdot \Delta t$ C_L: Cursor links C_R: Cursor rechts n: Anzahl Samples y_i: Y-Wert an Position i Δt: Abtastintervall
Rectified Mean Periodic	Sucht die Durchgänge der Basislinie und berechnet den gleichgerichteten Mittelwert des Signals von ganzen Perioden. $Rect_{Periodic} = \frac{1}{n \cdot \Delta t} \cdot \sum_{i=LC_L}^{LC_R} \lvert y_i \rvert \cdot \Delta t$ LC_L: Level Crossing links LC_R: Level Crossing rechts n: Anzahl Samples y_i: Y-Wert an Position i Δt: Abtastintervall

Rightmost Value	Zeigt den letzten Wert an der aktuellen Kurve im Datenerfassungsspeicher. Diese Funktion eignet sich, um im Kontinuierlichen Mode den aktuellen Messwert anzuzeigen während die Aufnahme läuft. Diese Messgrösse ist unabhängig von den Cursoren. Diese Messgrösse wird auch bei laufender Erfassung angezeigt.
RMS	Berechnet den quadratischen Mittelwert. $RMS = \sqrt{\frac{1}{n \cdot \Delta t} \cdot \sum_{i=C_L}^{C_R} y_i^2 \cdot \Delta t}$ C_L: Cursor links C_R: Cursor rechts n: Anzahl Samples y_i: Y-Wert an Position i Δt: Abtastintervall
RMS Periodic	Sucht die Durchgänge der Basislinie und berechnet den Effektivwert der ganzen Perioden. $RMS_{Periodic} = \sqrt{\frac{1}{n \cdot \Delta t} \cdot \sum_{i=LC_L}^{LC_R} y_i^2 \cdot \Delta t}$ LC_L: Level crossing links LC_R: Level crossing rechts n: Anzahl Samples y_i: Y-Wert an Position i Δt: Abtastintervall
StdDev	Berechnet die Standardabweichung des Signals. Die Standardabweichung entspricht einer RMS-Berechnung mit Abzug des Offsets. $SDev = \sqrt{\frac{1}{n \cdot \Delta t} \cdot \sum_{i=C_L}^{C_R} (y_i - mean)^2 \cdot \Delta t}$ C_L: Cursor links C_R: Cursor rechts n: Anzahl Samples

	y_i: Y-Wert an Position i Δt: Abtastintervall
StdDev Periodic	Sucht die Durchgänge der Basislinie und berechnet die Standardabweichung des Signals von ganzen Perioden. Die periodische Standardabweichung entspricht einer periodischen RMS-Berechnung mit Abzug des Offsets. $SDev_{Periodic} = \sqrt{\frac{1}{n \cdot \Delta t} \cdot \sum_{i=LC_L}^{LC_R} (y_i - mean)^2 \cdot \Delta t}$ LC_L: Level Crossing links LC_R: Level Crossing rechts n: Anzahl Samples y_i: Y-Wert an Position i Δt: Abtastintervall

36 Diverses

Weitere Informationen und Beschreibung von Funktionen und Zubehör:

36.1 ActiveX/COM - Interface

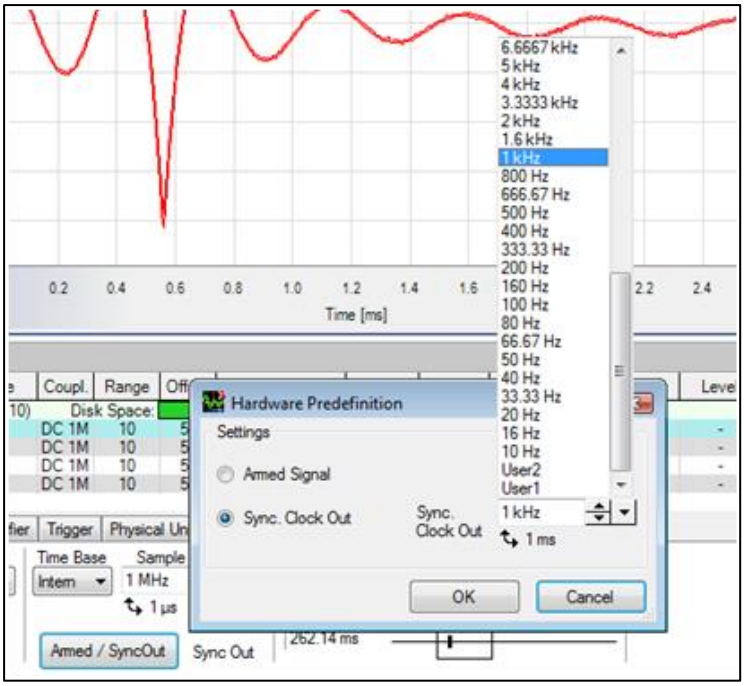
Die Active-X/COM - Schnittstelle ist als Option zu TranAX erhältlich.

Die COM (Component Object Model), aus von Microsoft, ist eine Möglichkeit, wie Komponenten (Programme) untereinander kommunizieren können. Diese Möglichkeit wird verwendet, um über Drittprogramme (Bsp. BallAX oder Excel Visual-Basic) TranAX zu steuern.

Die genaue Verwendung und Einstellungen der Parameter können dem ActiveX-Handbuch entnommen werden.

36.2 Sync.Clock Out

Ab Firmware Version 4 des Sternverteilers steht die Möglichkeit des **Sync.Clock Out** zur Verfügung. Diese Funktion kann dazu verwendet werden, **externe Geräte,** wie z.B. eine Highspeed Kamera zu **synchronisieren**.



Die Frequenz des Sync.Clock-Ausgangs kann aus einer **Dropdownliste** ausgewählt oder **manuell** eingegeben werden. Der eingegeben Wert wird dann auf den nächsten passenden gerundet.
Es stehen Frequenzen zwischen **10Hz und 10MHz** zur Verfügung. Die beiden Einträge in der Liste, "User 1" und "User 2" sind reserviert und können vor der Auslieferung) auf **Frequenzen gemäss Kundenwunsch programmiert** werden.
Beim Karteireiter ***"Haupt-Einstellungen"*** im **Control Panel** kann durch Klicken des Knopfs ***"Armed/SyncOut"*** der Dialog ***"Hardware Predefinition"*** geöffnet und zwischen **Armed Signal** und **Sync. Clock Out** gewählt werden. Normalerweise wird das Sync.Clock- bzw. das Armed-Signal am 25pol. D-Sub-Stecker ausgegeben (siehe Hardware-Beschreibung).

Um die **Sync.Clock Out** Funktion zu verwenden, muss die Installierte Software eventuell aktualisiert werden. Die folgenden Versionen werden als **Minimum vorausgesetzt**:

TranAX:	**3.2.1.702**	(Menü ***"Hilfe" / "Info"***)
TPC-Server:	**1.3.5**	(Control Panel /)
Sternverteiler:	**4**	(TraNetConfiguration.exe / Show Logfile / Server)

Ein Update der Firmware des Sternverteilers muss im Lieferwerk erfolgen!

Bei hohen Frequenzen (über 1MHz) muss auf die **Länge des Kabels** geachtet werden (< 2m bei 10MHz). Ansonsten sind die Signale beim Endgerät nicht mehr zuverlässig verwendbar.

Die Frequenzen für "User 1" und "User 2" müssen mit einer **angepassten Firmware kundenspezifisch einprogrammiert** werden und können nicht ad hoc geändert werden. **Für Änderungen (wie auch bei Updates von Geräten, die vor 2012 ausgeliefert wurden) muss der Sternverteiler zurück ans Lieferwerk gesendet werden.**

36.3 Konsolen Parameter

TranAX lässt sich auch aus der Konsole oder einer Batch-Datei starten. Die vorhandenen Parameter bieten dem routinierten Anwender eine Fülle von Optionen und Möglichkeiten.

Diese Beschreibung ist für **fortgeschrittene Anwender** und setzt ein bestimmtes **Grundwissen** im Umgang mit der Konsole und Batch-Dateien voraus. Es wird nicht weiter in die Details der jeweiligen Windows Versionen eingegangen. **Bei Fragen wenden Sie sich bitte an Ihre interne IT-Abteilung.**

Um die möglichen Parameter in der Konsole aufzulisten, wechseln Sie in das TranAX-Verzeichnis (in der Regel "C:\Program Files\Elsys\TranAX_x.x") und geben Sie ***TranAX.exe -help*** ein.

Die untenstehende Liste zeigt, die die Ausgabe und verfügbaren Parameter diese werden unabhängig der eingestellten Sprache immer in Englisch ausgegeben:

Parameter	Description and example
-allsettings	Loads all settings from the specific path. If the file name or directory path has blanks you have to write the whole path with quotes EXAMPLE: TranAX.exe -allsettings="C:\YOURPATH\MyAllSettings.lay"
-autoseq	Loads the autosequence from the specific path and starts it. If the file name or directory path has blanks you have to write the whole path with quotes EXAMPLE: TranAX.exe -autoseq="C:\YOURPATH\MyAutoSeq.aut"
-cachesize	Limits the data cache to the specified amount of bytes (default 150'000'000) EXAMPLE: TranAX.exe -cachesize=200000000
-device	Sets url of device(s) to use EXAMPLE: TranAX.exe -device=192.168.0.102:10010
-experiment	Opens the given Experiment. If the file name or directory path has blanks you have to write the whole path with quotes EXAMPLE: TranAX.exe -experiment="C:\YOURPATH\Experiment.exp"
-experimentSet	Opens the given experiment set. If the file name or directory path has blanks you have to write the whole path with quotes EXAMPLE: TranAX.exe -experiment="C:\YOURPATH\Experiment.exp\Experiment.zip"

-formula	Loads the formula file from the specific path. If the file name or directory path has blanks you have to write the whole path with quotes EXAMPLE: TranAX.exe -formula="C:\YOURPATH\MyFormula.for"
-help	Shows all commands EXAMPLE: TranAX.exe -help
-info	Shows metadata of a TPC5-File. If the file name or directory path has blanks you have to write the whole path with quotes EXAMPLE: TranAX.exe -info=C:\YOURPATH\test.tpc5
-layout	Loads the layout from a specific path. If the file name or directory path has blanks you have to write the whole path with quotes EXAMPLE: TranAX.exe -layout="C:\YOURPATH\MyLayout.lay"
-recording	Loads the recording settings from the specific path. If the file name or directory path has blanks you have to write the whole path with quotes EXAMPLE: TranAX.exe -recording="C:\YOURPATH\MyRecording.tps.xml"
-server	Acts as remote server (on port 12668) EXAMPLE: TranAX.exe -server
-serverport	Acts as remote server with a defined port EXAMPLE: TranAX.exe -serverport=12000
-title	Sets the title of the TranAX program. This could be necessary if you open several TranAX program EXAMPLE: TranAX.exe -title=VoltageMeasurement
-version	Shows the current build version of TranAX EXAMPLE: TranAX.exe -version
-view	Adds a file to the signal source browser. If the file name or directory path has blanks you have to write the whole path with quotes EXAMPLE: TranAX.exe -view="C:\YOURPATH\test.tpc5"
-offline	Starts TranAX without connecting to a device EXAMPLE: TranAX.exe –offline
-viewer	Starts TranAX as a viewer. The user can not manipulate measurement configurations (Start, stop, etc.) EXAMPLE: TranAX.exe -viewer
-scope	Starts TranAX in fullscreen mode with a Scope-Display EXAMPLE: TranAX.exe -scope
-fullscreen	Starts TranAX in fullscreen mode EXAMPLE: TranAX.exe -fullscreen

36.4 Verknüpfungen erstellen

Sie können TranAX-Experimente auch mit (Desktop)-Verknüpfungen oder einem Batchfile starten. Um TranAX direkt mit einem bestehenden Experiment zu laden, fügen Sie der Kommandozeile das folgende Kommando hinzu: ***-experimentset="experimentset name"*** Falls noch zusätzlich eine Autosequenz gestartet werden soll, geben Sie den entsprechenden Namen an: ***-autosequence="autosequence name"***

Beispieldatei "startTranAX.bat":

```
set TRANAX="C:\Program Files\Elsys\TranAX_4.1\TranAX.exe"
set EXPSET=".\myExperiment.exp\myExperimentSet.zip"

start "TranAX" %TRANAX% -experimentset=%EXPSET%
```

36.5 Einschränkungen

Die folgenden Punkte behandeln gewisse Einschränkungen, die bei den Einstellungen beachtet werden müssen.

36.5.1 Digitale Eingänge (Marker)

Digitale Eingänge sind nur dann verfügbar, wenn optional die entsprechende **Marker-Hardware** installiert ist. Ein Marker-Signal steht immer in Bezug zum Namen eines Analog-Kanals. Jeder mit der Option ausgestattete Analog-Kanal enthält 2 Marker-Spuren. Die Software kann keine direkten Hinweise liefern bei welchen Kanälen Marker installiert sind. Bei 16-Bit Modulen können Marker nur aufgezeichnet werden, wenn der ADC auf 14-Bit gestellt wird (siehe Mittelung). Marker-Kanäle können nicht als Trigger verwendet werden, aber es kann ein externer digitaler Trigger-Eingang pro Gerät verwendet werden. Der externe Trigger wird via Trigger-Einstellungen gesteuert.

36.5.2 Differential-Eingänge

Mit den Single-Ended Modulen (z.B. TPCX-2014-**8S**) werden die folgenden Kanal-Paarungen für die **differentiellen Eingänge** verwendet: (1,2), (3,4), (5,6), (7,8). In diesem Fall sind die **geradzahligen Kanäle** nicht einstellbar.
Mit den differentiellen Modulen (z.B. TPCX-2014-**8D**) sind alle Kanäle mit zwei BNC-Steckern versehen, deshalb haben auch geradzahlige Kanäle differentielle Eingänge.
Die entsprechenden Einstellungen werden im Abschnitt Eingangs-Verstärker beschrieben.

36.5.3 Maximale Abtastrate

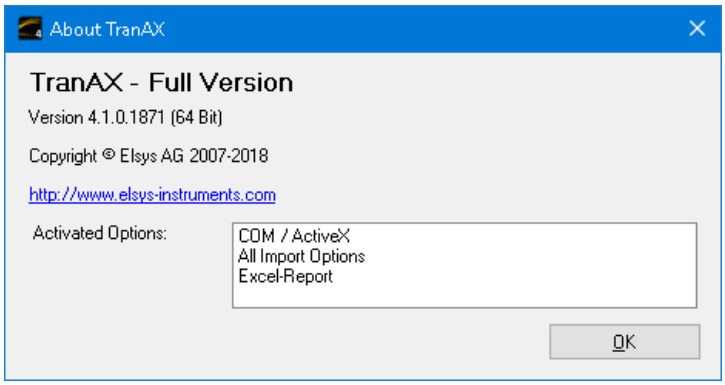
Die **vertikale Auflösung von 16-Bit** (mit den 16-Bit-Modulen wie TPCX-4016-4D) ist nur gegeben bei **Abtastraten gleich oder unterhalb 1/4 der maximalen Abtastrate**. Beispiel: Bei einer TPCX-4016-4x Modul (40MHz), ist die 16-Bit-Auflösung nur für Abtastraten gleich oder kleiner 10 MS/s gegeben. Oberhalb von 10 MS/s bis 40 MS/s entspricht die vertikale Auflösung dem 14-Bit ADC.

37 Fehlerbehebung

Um Fehlverhalten schnell zu lösen ist es entscheidend, dass der Support mit den wichtigsten Informationen versorgt wird. Dieses Kapitel beschreibt wie Probleme selbst gelöst werden können und welche Daten für allfällige Anfragen mitgegeben werden sollten.

37.1 TranAX Software Version

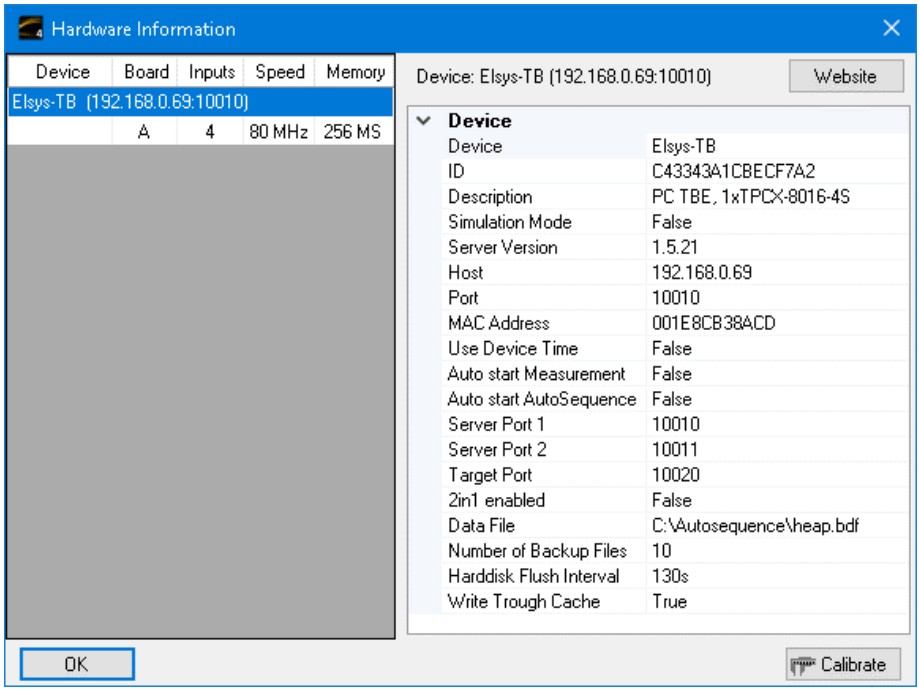
Um eine Messung in TranAX zu erfassen sind mehrere Software Komponenten miteinander in Verbindung. TranAX kommuniziert mit dem TPC-Server, dieser mit dem Treiber und dieser wieder mit den TPCX/TPCE Modulen.



Um die Version Ihrer TranAX Version zu bestimmen, klicken Sie im Menü ***"Hilfe" / "Info"***. In diesem Beispiel handelt es sich um die Version 4.1.1871.

37.2 TPC-Server Version

Die Version des TPC-Servers kann wie folgt bestimmt werden: Klicken Sie im Control Panel auf den Informationsknopf , anschliessend wir das folgenden Fenster geöffnet. Hier ist die Server Version 1.5.21 installiert.

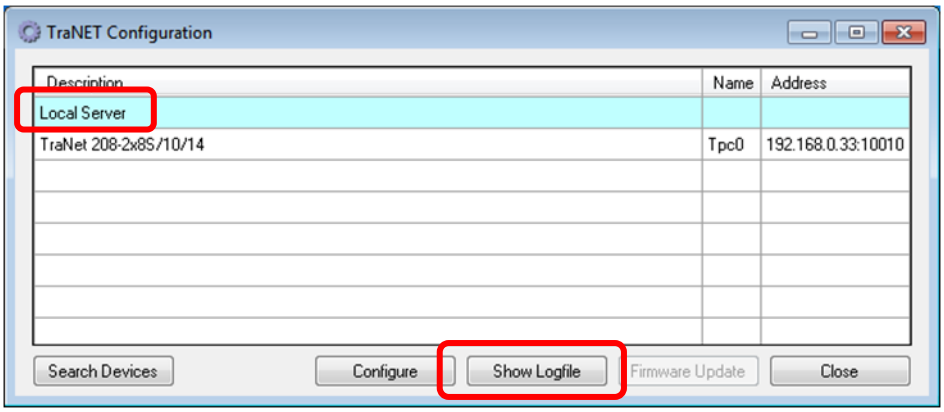


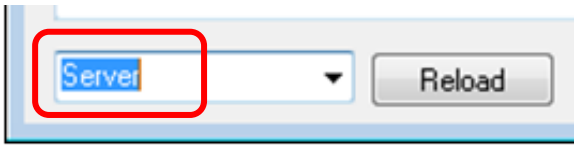
37.3 Treiber und Firmware Version

Um die Version der Treiber, der Firmware für die TPCX/TPCE -Module und des Sternverteilers zu ermitteln, muss erst TraNetConfiguration.exe gestartet werden. Dazu doppel klicken Sie das folgende Symbol auf dem Desktop.

Anschliessend wählen Sie im "TraNET Configuration" Dialog Ihr Gerät, entweder "Local Server" oder eines der gefunden TraNET FE Geräte, aus. Anschliessend klicken Sie auf den Show Logfile Knopf.

Es werden nur TraNET FE und das Lokale Device gelistet. Weitere TraNET EPC und PPC Geräte müssen auf den jeweiligen Rechnern direkt kontrolliert werden.





Achten Sie darauf, dass im Dropdown-Menü **"Server"** ausgewählt ist. Im Textfenster finden Sie nun **alle Informationen** zu Ihrem TraNET System.

Ein **Update des TPC-Servers** beinhaltet die entsprechende Version des **TPC-Server** selbst, der **Firmware für die TPCX/TPCE Module** und des **Windows Treibers**.
Ein Update für TranAX beinhaltet lediglich die Bedien-Software TranAX.

Bei Fragen und Problemen speichern Sie die Server-Einstellungen durch Klicken des Knopfs "Save" und senden Sie diese Datei anschliessend zusammen mit Ihrer Fehlerbeschreibung an die zuständige Support-Stelle.

Es wird generell empfohlen, die **gesamte Software gemäss Installation CD zu installieren,** um allfälligen Inkompatibilitäten zwischen den Software Versionen vorzubeugen.

37.3.1 Beispiel Windows 7

```
Server number 0 started
==========================================================
Operation System: Microsoft Windows 7
Server version: 10305

==========================================================
Board number 0 found
Driver Version 2.3.3 found
I2C Delay Factor: 4
Firmware "Version A.1.6.2 Overload Reset changed" found
No Synchronisation found
```

In diesem Fall ist ein Windows 7 System im Einsatz mit derselben Software wie oben beim Beispiel mit Windows XP. Die Meldung **"No Synchronisation found"** deutet darauf hin, dass entweder ist **kein Sternverteiler vorhanden** ist, dieser **nicht richtig angeschlossen** ist oder **defekt ist**.

37.3.2 TraNET FE

```
Server number 0 started
==========================================================
Model Type: TraNET-FE
Operation System: Linux
Server version: 10305

==========================================================
Board number 0 found
Driver Version 2.3.2 found
I2C Delay Factor: 10
Firmware "Version A.1.6.2 Overload Reset changed" found
Starhub Version 1 found
```

Hier steht oben zusätzlich **Model Type: TraNET FE**. Als Betriebssystem kommt hier in **Linux** zum Einsatz. Die Übrigen Informationen sind identisch, ausser dass hier **Sternverteiler** die Firmware **Version 1** hat.

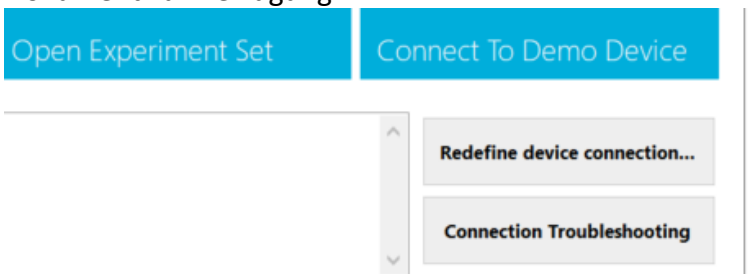
37.4 Fehlermeldungen

37.4.1 TranAX

In TranAX können die folgenden Fehlermeldungen im Control Panel angezeigt werden:

Meldung	Ursache	Lösung
"Hardware failure"	Kein TPCX/TPCE Modul vorhanden	TPCX/TPXE installieren oder TPC-Server in Demo Mode umschalten.
	TPCX/TPCE- Modul nicht richtig eingebaut	Bitte kontrollieren Sie, dass die PCI/PCIe Karte richtig eingebaut und fest im Steckplatz eingebaut ist. Der Rechner muss dazu ausgeschaltet werden!
	Treiber nicht richtig installiert	TPC-Server erneut installieren. Dieser wird mit der CD im Handbuch mitgeliefert.
	PCI/PCIe Steckplatz defekt	Kontrollieren Sie, ab durch einen Wechsel des PCI/PCIe Steckplatzes der Fehler immer noch auftaucht.
	TPCX/TPCE -Modul defekt	Falls keine der oberen Möglichkeiten hilft, muss das TPCX/TPCE-Modul zur genaueren Analyse eingesendet werden.
"Network Error"	Netzwerkkabel nicht angeschlossen	Netzwerkkabel anschliessen, je nach Verbindung muss ein gekreuztes verwendet werden
	TraNET FE oder EPC nicht gestartet	Am Strom anschliessen und starten, wird durch die grün blinkende LED an der Front signalisiert
	Falsche IP Adresse	Kontrollieren, ob im TranAX auch das richtige Gerät oder die lokale Adresse angewählt wurde.
	Falscher Port	Kontrolle ob der richtige Port eingestellt wurde. TranAX muss auf den Port verbinden, der das TraNET zur Verfügung stellt, Kontrolle im TraNET Config.
	Netzwerk Kollisionen	Sicherstellen, dass jedes Gerät eine eigene IP-Adresse besitzt, die von keinem anderen Gerät im lokalen Netzwerk verwendet wird. Eventuell den internen IT-Support kontaktieren, dass eine Adresse zugewiesen wird.

Auf der Startseite von TranAX steht ein Handbuch zur Fehlerbehebung bei Verbindungen als PDF-Dokument zur Verfügung.



37.4.2 TraNET Config Logfile

Weitere wichtige Informationen zum Ihrem Messystem können Sie im Logfile des Servers von TraNET Config entnehmen.

Meldung	Ursache	Lösung
"No Synchronisation found"	Es ist kein Sternverteiler eingebaut	Kein Fehler
	Sternverteiler nicht angeschlossen	Kontrolle der Verbindungskabel zwischen Sternverteiler und TPCX/TPCE-Modulen.
	Sternverteiler defekt	Falls keine der oberen Möglichkeiten hilft, muss der Sternverteile zur genaueren Analyse eingesendet werden.
Keine Einträge vorhanden *"Board number x found"*	Kein TPCX/TPCE Modul vorhanden	Modul erwerben oder TPC-Server in Demo Mode umschalten.
	TPCX/TPCE - Modul nicht richtig eingebaut	Bitte kontrollieren Sie, dass die PCI/PCIe Karte richtig eingebaut und fest im Steckplatz eingebaut ist. Der Rechner muss dazu ausgeschaltet werden!
	Treiber nicht richtig installiert	TPC-Server erneut installieren. Dieser wird mit der CD im Handbuch mitgeliefert.
	PCI/PCIe Steckplatz defekt	Kontrollieren Sie, ab durch einen Wechsel des PCI/PCIe Steckplatzes der Fehler immer noch auftaucht.
	TPCX/TPCE -Modul defekt	Falls keine der oberen Möglichkeiten hilft, muss das TPCX/TPCE -Modul zur genaueren Analyse eingesendet werden.
Keine Einträge vorhanden *"Server number 0 started" fehlt*	TPC-Server nicht installiert oder fehlerhafte Installation	Installieren Sie TPC-Server mit der mitgelieferten CD neu, Eventuell ist ein Neustart des Systems notwendig.

37.4.3 TranAX Firewall und Port Einstellungen

TranAX kommuniziert per TPC/IP mit dem TPCServer. Dieser läuft entweder lokal auf Windows System beim EPC und PPC oder auch bei externen Geräten wie dem TraNET FE.

- Der Deivce Finder von TranAX verwendet den Port 10020 UDP um nach Geräten zu suchen.
- TranAX verwendet den Port 10010 TPC um mit den Geräten zu kommunizieren und Daten zu holen.

Stellen Sie sicher, dass der **Port 10020 UPD** und der **Port 10010 TPC** für TranAX in **beide Richtungen** freigegeben ist. Die Windows Standard Firewall werden diese Einstellungen automatisch vorgenommen.

Dokumentinformationen

Stand TranAX	4.1.2.2320
Dateiname:	TranAX_4.1.2_Manual_DE.docx
Status:	Freigegeben

Letzte Änderungen

Datum:	20.05.2020
Autor:	MM/TBE