MathScript Dokumentation

Zur Verwendung mit GSVmultichannel ab Version 1.48 (MathScript Version 1)

Änderungsnachweis		
Version	**Status**	**Bearbeiter**
1.00	initiiert	SW
1.01	typo & sprachl Korrekturen	SW

ME-Meßsysteme GmbH
Eduard-Maurer Str. 9
16761 Hennigsdorf
Tel.: +49 3302 78620 60
Fax: +49 3302 78620 69
Mail: info@me-systeme.de
Web: www.me-systeme.de

Inhaltsverzeichnis

ME-Meßsysteme GmbH
Eduard-Maurer Str. 9
16761 Hennigsdorf
Tel.: +49 3302 78620 60
Fax: +49 3302 78620 69
Mail: info@me-systeme.de
Web: www.me-systeme.de

Einführung

MathScript ist eine Datei mit Beschreibungen von Berechnungen, mit denen MathChannels erzeugt werden können. Das sind Software-Kanäle, die in GSVmultichannel wie alle anderen Kanäle angezeigt und aufgezeichnet werden können.

Man kann damit Eingangswerte, das sind i.d.R. echte Hardware-Kanäle, umrechnen und auch Signale erzeugen, d.h. Funktionen nach der Zeit f(t). MathScript kann zu letzterem Zweck auch ohne angeschlossene Hardware betrieben werden. Das ist beispielsweise nützlich, um Berechnungen mit definierten Eingangssignalen zu testen.

Verwendung:

Wenn MathScript zugleich mit angeschlossener Hardware (Messverstärker) verwendet werden soll, müssen mit Add Channel erst alle Hardwarekanäle geöffnet werden.
Anschließend wird im Add Channel Dialog der Devicetype „MathScript“ gewählt. Es öffnet sich ein Dateiauswahldialog, mit dem das gewünschte Skript ausgewählt wird.
Anschließend wählt man unter „Input Channel“ die Ergebniskanäle des Skripts (=linksseitige Variablen x<no>, s.u.) aus, die in GSVmulti angezeigt werden sollen. Man kann dabei einzelne auswählen oder Bereiche oder alle. Auch bei MathChannels wird der Add Channel Dialog mit Connect bestätigt.
Anschließend wertet GSVmulti das Skript aus. Tritt dabei ein Fehler auf, wird dies gemeldet, andernfalls wird das Skript in ein optimiertes internes Format übersetzt, das zur Messlaufzeit verwendet wird.
Wenn alle MathChannels korrekt eingebunden werden konnten, können sie wie andere Kanäle verwendet werden; allerdings steht weniger Parametrierung zur Verfügung, so dass im Configuration Tab einige Buttons ausgegraut sind.

Die Kanalkonfiguration kann mit Save Session gespeichert und mit Load Session später wieder hergestellt werden; dabei ist es wichtig, dass das MathScript mit dem gleichem Dateinamen im gleichen Pfad verbleibt (weil nur dieser in der Sessiondatei gespeichert wird, nicht aber der Skript-Inhalt an sich).

Zur Laufzeit wird das Skript zu jeden Einzelmesswert aller Kanäle einmal durchlaufen.

Den MathChannels kann eine beliebige Einheit zugeordnet werden; diese wird in der Session-Datei mit abgespeichert, d.h. mit Load Session wieder hergestellt. Das Wählen der Einheit erfolgt wie bei allen Kanälen mit dem Drop-Down-Menü im Configuration Tab oder per Menüleiste→Channel→Unit.

Wenn gewünscht, können Kanäle aus Messverstärkern in der Ansicht versteckt werden (Channel→Hide). Das ist nützlich, wenn dieser Kanal einen Eingangswert für MathScript

darstellt, aber nicht angezeigt werden soll.

Es kann nur ein MathScript verwendet werden, d.h. dieses muss die Berechnung aller gewünschten MathChannels enthalten.

Erstellen des MathScripts

1. Form

MathScript ist eine Textdatei mit der Endung txt.
Die Zeichenkodierung ist ASCII only, d.h. sie darf keine Sonderzeichen enthalten, die nicht Teil des ASCII Zeichensatzes sind (mit Ausnahme der Kommentartexte, die von der Software ignoriert werden).

Jedes Skript besteht mindestens aus Anweisungen (mindestens einer) und optional aus Kommentaren und globalen Parametern. Bei Anweisungen und globalen Parametern ist die Groß- und Kleinschreibung zu beachten.

Das Zeilenendzeichen darf Windows- oder Linux-Form haben.

1.1. Anweisungen

Anweisungen werden immer mit dem Semikolon abgeschlossen. Es dürfen zwischen Statements (d.h. Variablen und Schlüsselwörtern) beliebig viele Leerzeichen, Tabulator-Zeichen und Zeilenendzeichen stehen.

Genaueres dazu in Punkt 2.

1.2. Kommentare (optional)

Kommentare müssen als erstes Zeichen einer Zeile das # Zeichen (Kardinal/"Hash-Tag") haben. Gehen sie über mehrere Zeilen, muss in jeder Zeile an Anfang wieder # stehen.

1.3. Globale Parameter (optional)

Globale Parameter haben die Form

_<Name> = <Wert>

Alle diese Bezeichner beginnen also mit einem Unterstrich (Underscore).
Sie sollten am Anfang einer Zeile stehen und haben kein Zeilenendzeichen.

Es gibt z. Zt. (Version 1) 4 verschiedene Parameter:

1.3.1.

_ScriptNo = <no>

Diese Nummer dient nur zur Unterscheidung verschiedenen Skripte für den Benutzer. Die Nummer wird im Configuration-Tab als "Serial No." angezeigt, wenn ein MathChannel gewählt ist. Wenn der Eintrag _ScriptNo fehlt, wird 0 zugeordnet.

1.3.2.

_Version = <no>

Hiermit können künftige Erweiterungen unterschieden werden. Fehlt _Version, wird Version 1 angenommen.

1.3.3.

_DataFreq = <wert>

Die angegebene Datenrate gilt, wenn MathScript ohne Hardware verwendet wird, nach Start des MathScripts.

Sie bezeichnet die Anzahl der Durchläufe pro Sekunde, d.h. die Anzahl berechneter MathChannel-Arrays pro Sekunde. Der Wertebereich ist 1 bis 1000 (allerdings werden hohe Datenfrequenzen nicht genau erzeugt; je nach PC-Leistung, Systemauslastung und MathScript-Komplexität ist die erreichbare Datenfrequenz oft nur ca. 300/s).

Die Datenrate kann mit GSVmulti geändert werden, wenn keine Hardware angeschlossen ist; allerdings ist diese Änderung flüchtig, d.h. <wert> wird nicht automatisch in MathScript eingetragen.

Fehlt _DataFreq, wird der Defaultwert 10/s verwendet.

Wenn Messverstärker angeschlossen sind, so gilt die Datenrate der Geräte und _DataFreq wird ignoriert.

Es ist (aus vielen Gründen) empfehlenswert, bei mehreren eingebundenen Geräten diese alle möglichst auf die gleiche Datenfrequenz einzustellen.

1.3.4.

_InArrayNo = <no>

Wenn GSVmulti unabhängig von MathScript auch Sondersensor-Berechnungen durchführt, z.B. Spannungsmessung bei Rosetten-DMS oder für Mehrachsen-Sensoren, kann man hiermit angeben, ob die Eingangswerte in ch<no> diese bereits berechneten (physikalischen) Werte darstellen oder Rohwerte. Bei Rosetten sind die Rohwerte Dehnungen in der Einheit µm/m und bei Mehrachsensensoren Brückenverstimmungen in

mV/V.

Mit <no> =0 wird angegeben, dass die Eingangswerte berechnete Werte sind (default) oder Rohwerte mit <no> =1.

Fehlt der Parameter _InArrayNo, wird 0 angenommen, d.h. berechnete Werte.

Wenn keine Software-berechneten Sondersensoren verwendet werden (z.B. auch mit GSV-6/-8 und hardware- berechnetem Sechsachsensensor), hat dieser Parameter keine Bedeutung.

2. Erstellen der Anweisungen

Anweisungen sind in reine arithmetische Ausdrücke, Definitionen von Konstanten und Programmieranweisungen kategorisiert und sie können Operatoren, Schlüsselwörter, Variablen und numerische Konstanten enthalten. Variablen bezeichnen Ein- und Ausgangswerte.

2.1. Arithmetische Ausdrücke

Nicht-bedingte arithmetische Ausdrücke haben stets die Form

x<no>= <Ausdruck>;

x<no> ist die festgelegte Form der linksseitigen Variablen; <no> geht von 1 bis 99, d.h. es können bis zu 99 Ausdrücke definiert werden, s. 2.2.2.

Der <Ausdruck> selbst darf auch linksseitige Variablen enthalten, sofern diese vorher definiert wurden. Er enthält Variablen (siehe 2.2.), numerische Konstanten (s. 2.3.) und Operatoren. Operatoren sind solche der Grundrechenarten und arithmetische Funktionen (siehe Anhang). Alle arithmetischen Funktionen haben genau ein Argument. Die Grundoperatoren werden in normaler mathematischer Vorwärts-Reihenfolge verwendet. Sie umfassen:

+ <Addition>
- <Subtraktion>
* <Multiplikation>
/ <Division>
^ <Potenz>

Terme im Ausdruck dürfen mit runden Klammern gegliedert werden. Die Präferenz der Multiplikation und Division vor Addition und Subtraktion ist MathScript bekannt; bei sehr komplexen Ausdrücken wird dennoch die Verwendung von Klammern empfohlen.

Alle Berechnungen erfolgen intern mit dem Datenformat 64-Bit Double.

Beispiel 1:

Multipliziere die Werte von Kanal 1 (aus Hardware) mit 0,3 und addiere zum Produkt 5 und weise
das Ergebnis der Variablen x1 zu:
x1= 5 + ch1*0,3;

Die linksseitigen Variablen sind mit allen ihren rechtsseitigen Ausdrücken (auch bedingten) in aufsteigender Reihenfolge zu verwenden, d.h. die erste ist x1, dann folgt x2 (wenn vorhanden) usw.

2.2. Variablen

Variablen haben eine festgelegte Schreibweise mit festgelegter Bedeutung. Die Gesamt-Anzahl verwendeter Variablen ist auf 280 beschränkt; zu weiteren Einschränkungen siehe Kapitel 3.

2.2.1. Aktuelle Hardware-Eingangswerte

Sie bezeichnen Messwerte des aktuellen Durchlaufs. Die Bezeichnung ist:

ch<no> ch1 … ch99

<no> bezieht sich auf die Kanal-Nummer in GSVmultichannel, die in "Actual Channel" angezeigt wird und darf von 1 bis 99 gehen. Der Eingangskanal muss vorhanden sein, d.h. vorher mit Add Channel geöffnet worden oder Teil der Sessiondatei sein. Eingangswerte dürfen nur rechtsseitig verwendet, d.h. nur gelesen werden.

2.2.2. Ausgangswerte

Sie bezeichnen linksseitige Ergebnisse von Berechnungen. Sie können daher mit Add Channel geöffnet werden, müssen es aber nicht, d.h. es darf sich auch um Zwischenwerte handeln, die nicht angezeigt werden. Definierte Werte dürfen in anschließend folgenden Ausdrücken auch rechtsseitig verwendet, d.h. geschrieben und gelesen werden .Die Bezeichnung ist:

x<no> x1 … x99

<no> ist eine fortlaufende Nummer, beginnend mit 1, dann 2, usw., bis 99.

Beispiel 2:

x1= ch1*0,3 + 5;
Berechne ch7 hoch (ch1*0,3 + 5) und weise das Ergebnis x2 zu
x2= ch7^x1;

2.2.3. Eingangswerte aus dem letzten Durchlauf

Bei jedem Berechnungsdurchlauf, d.h. zu jedem Zeitpunkt, zu dem das Array der Eingangskanalwerte erfasst wurde, wird MathScript einmal ausgeführt und dabei werden die Eingangswerte ch<no> gespeichert. Es kann auf die Eingangswerte aus den vorherigen Durchlauf zugegriffen werden, und zwar mit:

ch<no>L ch1L … ch99L

Auch die Eingangswerte des letzten Durchlaufs dürfen nur rechtsseitig verwendet, d.h. nur gelesen werden.

Beispiel 3:

```
# Bilde die diskrete Steigung, d.h. die Differenz zum letzten Messwert des Kanals 1 und weise
# dieses Ergebnis (das zeitlich unkorrelierte Differential dx) der Variablen x1 zu:
x1= ch1 – ch1L;
```

2.2.4. Ausgangswerte aus dem letzten Durchlauf

Bei jedem Berechnungsdurchlauf werden auch die Ausgangswerte x<no> gespeichert. Auf diese Werte aus dem letzten Durchlauf kann zugegriffen werden mit

x<no>L x1L … x99L

Auch die Ausgangswerte des letzten Durchlaufs dürfen nur rechtsseitig verwendet, d.h. nur gelesen werden. <no> entspricht der des x-Wertes, z.B. ist x1L der letzte Wert von x1. x<no> muss also in dem Skript definiert sein.

Beispiel 4:

```
# Erzeuge ein Sinussignal (zur Bedeutung von CNT: s. 2.2.5.)
x1= sin(CNT*6,2832/100);
# Bilde die Ableitung aus dem Sinus x1, d.h. ein Cosinus-Signal mit derselben Amplitude 1:
x2= (100/6,2832) * (x1-x1L);
```

2.2.5. Sonderwerte

Es gibt z. Zt. (Version 1) zwei Sondervariablen:

CNT und FREQ.

CNT ist ein laufender ganzzahliger Zähler, der bei einem Reset-Ereignis (siehe 3.) mit 0 initialisiert und in jedem Durchlauf nach Ausführung aller Anweisungen um 1 erhöht wird. Er kann programmatisch mit der RESET Anweisung zurückgesetzt werden (s. 2.3.4). Geschieht dies jedoch nie, so setzt er sich nach 4.294.967.295 (2^32 -1) Durchläufen selbst auf 0.
CNT ist u.a. geeignet, von der diskreten Zeit abhängige Funktionen f(t), d.h. Signale mit diskretisierter Zeit zu erzeugen, weil die Zeit zwischen zwei Durchläufen 1/FREQ, d.h. der Messdatenperiode entspricht (mathematisch wird ein solches Signal also mit y= f(kT) beschrieben, wobei k=CNT und T=1/FREQ ist). Zur Verrechnung wird CNT intern in eine Fließkommazahl umgewandelt.

Siehe Beispiele 4 und 6.

ME-Meßsysteme GmbH
Eduard-Maurer Str. 9
16761 Hennigsdorf
Tel.: +49 3302 78620 60
Fax: +49 3302 78620 69
Mail: info@me-systeme.de
Web: www.me-systeme.de

FREQ ist die aktuell eingestellte Messdatenfrequenz (Summenabtastrate) des Systems. Wenn mehrere Messverstärker mit unterschiedlichen Messdatenfrequenzen vorhanden sind, ist FREQ der Mittelwert aus maximaler und minimaler Messdatenrate (dieser Sonderfall wird allerdings nicht empfohlen).

Beispiel 5:

```
# Bilde von Kanal 1 die Integrale nach der Zeit über jede Messdatenperiode
x1= ch1 / FREQ;
```

2.3. Numerische Konstanten

Konstanten können als Ganzzahl, als Dezimalzahl mit Nachkommastellen oder als (wissenschaftliche) Exponentialdarstellung zur Basis 10 geschrieben werden. Der Dezimaltrenner entspricht i.d.R. dem der im Windows OS eingestellten Sprache. Ist dieser ein Komma (z.B. bei deutschem Windows), darf er ausnahmsweise auch . (period) sein. Numerische Konstanten dürfen in Ausdrücken oder Programmieranweisungen verwendet werden und müssen in der Definition von Konstanten vorhanden sein, und zwar rechts hinter dem Gleichheitszeichen =.
Beispiele (deutsch):
48 3,1415926 -5 2.598E5 -1,586E-3

2.4. Definition von Konstanten

Für numerische Konstanten können Ersatzbezeichner definiert werden. Diese dürfen in darauf folgenden Ausdrücken oder Programmieranweisungen verwendet werden. Die Definition hat die Form:

const<String>=<numerische Konstante>;

<String> steht direkt hinter *const* und darf alle Groß- und Kleinbuchstaben sowie Ziffern enthalten, aber keine Sonderzeichen wie z.B. _ ? . , * - <Leerzeichen>.

Beispiel 6:

```
constPI=3,1415926;
constE= 73000;
conste = 2.718281828;
const23 = -41.7E-6;
x1= ch1*constE / (constPI * (1+const23*conste));
```

In Beispiel 6 ergibt x1 mit ch1=1 das Ergebnis 23239,2559. Man beachte, dass *conste* und *constE* als verschieden behandelt werden (case-sensitive).

2.5 Programmieranweisungen

Alle Programmier-Schlüsselwörter werden in Großbuchstaben geschrieben. Auch Programmieranweisungen werden stets mit einem Semikolon abgeschlossen. Die Nomenklatur der Variablen (siehe 2.2.) ist die selbe wie bei arithmetischen Ausdrücken.
Es gibt 2 Kategorien von Anweisungen: Iterationen und Buffer-Anweisungen.

2.5.1. Iterationen

Iterationen sind Anweisungen, die nur ausgeführt werden, wenn eine Bedingung zutrifft. Sie können aus einer einzeln stehenden IF Anweisung bestehen oder einem IF / ELIF / ENDIF Block.
IF Anweisungen haben die Form:

IF(<Bedingung>) { <Anweisung>};

IF/ELIF/ENDIF-Blöcke haben die Form:

IF(<Bedingung>) { <Anweisung> };
ELIF(<Bedingung>) { <Anweisung> };
<optional weitere ELIF Anweisungen>
ENDIF;

Die Bedingung

In jeder IF / ELIF Anweisung kann nur eine Bedingung geprüft werden, eine boolsche Verknüpfung von mehreren Bedingungen ist also innerhalb einer Anweisung nicht möglich. Die Bedingung besteht aus einer Variablen und einer numerischen Konstante oder aus zwei Variablen, die miteinander verglichen werden.
Es existieren folgende 6 Vergleichsoperatoren:

>	<größer als>
<	<kleiner als>
>=	<größer oder gleich>
<=	<kleiner oder gleich>
=	<gleich>[1]
!=	<ungleich>

Die Bedingung in einer ELIF Anweisung wird nur dann geprüft, wenn alle vorhergehenden IF und ELIF Bedingungen nicht zutrafen. Sobald in einem IF/ELIF/ENDIF-Block eine Bedingung zutrifft, wird die entsprechende Anweisung ausgeführt und alle folgenden ELIF Bedingungen werden nicht mehr geprüft (ELIF entspricht also z.B. dem else if() Verhalten in

1 Der gleich/ungleich-Vergleich erfolgt im Intervall des sog. Machine Epsilon der 64-Bit Double Mantisse

der Programmiersprache C).
Wenn keine der Bedingungen zutrifft, wird keine Anweisung ausgeführt und der linksseitige Wert in der Anweisung x<no> behält den zuletzt zugewiesenen Wert, d.h. auch den des vorangehenden Skript-Durchlaufs, falls er im aktuellen Durchlauf noch nicht zugewiesen wurde.

Tipp: Um das Verhalten einer typischen ELSE Anweisung nachzubilden, also einer Anweisung, die ausgeführt wird, wenn keine der vorhergehenden IF/ELIF Bedingungen zutreffen, kann man die unmittelbar vorangehende IF/ELIF Bedingung mit den gleichen Argumenten logisch umkehren, siehe Beispiel 10.

Die Anweisung in IF / ELIF

Die Anweisung ist in geschweiften Klammern eingeschlossen. Mit Ausnahme des RESET Kommandos (siehe 2.5.2) bestehen Anweisungen aus genau einer Zuweisung der Form

x<no>= <Ausdruck>

Der Ausdruck darf genau eine arithmetische Anweisung sein oder eine Buffer-Anweisung.

Innerhalb eines IF/ELIF/ENDIF-Blocks müssen alle linksseitigen x<no> Variablen gleich sein (Version 1).

Beispiel 7:

```
# Skript ohne Hardware, erzeugt ein Dreiecksignal mit der Amplitude constA= 5
# Die Signalperiode wird in 3 Abschnitte unterteilt, d.h. abschnittsweise definiert
constA = 5;
IF(CNT < 25) {x1= CNT*4*constA/100};
ELIF(CNT < 75) {x1= 2*constA-(CNT*4*constA/100)};
ELIF(CNT < 100) {x1= (CNT*4*constA/100) - 4*constA};
ENDIF;
# Zurücksetzen des Counters. Wird nach der letzten ELIF Anweisung des Blocks oben getan,
# CNT hat also den Wertebereich 0 bis 99
IF(CNT=99) {RESET(0)};
```

2.5.2. Buffer-Anweisungen

Buffer Anweisungen ermöglichen das Aufsummieren oder eine Mittelwertbildung von zeitlich nacheinander folgenden Werten. Dabei werden interne Summenregister verwendet, die die Summen speichern. Es können bis zu 23 Summenregister verwendet werden.

SUM Anweisung

Mit der SUM Anweisung werden zeitlich aufeinander folgende Werte derselben Variable

summiert. Sie hat die Form:

x<no>= SUM<a>(<arg>, N);

Hierin bedeuten:

x<no>: Ergebnisvariable, linksseitiger Wert

<a>: Nummer des Summenregisters. Der Wertebereich ist 1 bis 23. Wird <a> weggelassen, wird 1 angenommen. Hiermit können z.B. verschiedene Argumente (Variablen) unabhängig voneinander summiert werden. Für jede SUM Anweisung ist dann ein neues <a> zu wählen, und zwar in aufsteigender Reihenfolge. Da die MEAN Anweisung dieselbe Summenregisterbank verwendet, sind bei der Wahl von <a> die MEAN Anweisungen mitzuzählen.

<arg>: Argument, d.h. Variable, die aufaddiert werden soll. Erlaubt sind: ch<no>, x<no> (sofern vorher definiert), ch<no>L, x<no>L (sofern x<no> definiert).

N: Anzahl der Werte, über die addiert werden soll. Wenn N>1, ist also jedes Ergebnis die Summe der (N-1)-letzten Werte plus dem aktuellen Wert[2] in <arg>. Dies gilt jedoch nur, wenn nach einem RESET-Ereignis bereits N oder mehr Werte vorhanden sind. Andernfalls ist das Ergebnis die Summe über alle vorhandenen Werte in <arg>.

Wenn N=0, wird immer weiter aufaddiert. Das Ergebnis ist also die Summe aller Werte in <arg> ab dem letzten RESET Ereignis (s.u.) bzw. RESET Kommando bis zum nächsten RESET Ereignis / Kommando.

Beispiel 8:

```
# Bilde die Summe über den aktuellen und die letzten 9 Werte des Kanals 1
x1= SUM1(ch1, 10);
# Bilde die Summe über den aktuellen und die letzten 99 Werte des Kanals 7
x2= SUM2(ch7, 100);
```

MEAN Anweisung

Mit der MEAN Anweisung wird der arithmetische Mittelwert von zeitlich aufeinander folgenden Werten derselben Variable gebildet. Sie hat die Form:

x<no>= MEAN<a>(<arg>,N);

Die Bedeutung der Platzhalter no, a, arg und N ist die gleiche wie bei der SUM Anweisung. Bei der Wahl von <a> wird mit 1 begonnen und dann weiter gezählt, einschließlich der SUM Anweisungen.
Der Unterschied zur SUM Anweisung ist, dass bei MEAN die Summe stets durch die Anzahl

2 Wenn <arg> einen Wert des letzten Durchlaufs bezeichnet, d.h. x<no>L oder ch<no>L, ist das Ergebnis die Summe der N-letzten Werte, ohne den aktuellen Wert.

ME-Meßsysteme GmbH
Eduard-Maurer Str. 9
16761 Hennigsdorf
Tel.: +49 3302 78620 60
Fax: +49 3302 78620 69
Mail: info@me-systeme.de
Web: www.me-systeme.de

aufsummierter Werte geteilt wird. Wenn N>1, ist dies N, falls seit dem RESET Ereignis bereits N Werte vorhanden sind; andernfalls die Anzahl vorhandener (=aufsummierter) Werte. Ist N=0, wird durch die Anzahl aufsummierter Werte geteilt und der Divisor wird immer größer - bis zu einem neuen RESET Ereignis.
Dadurch ergibt sich eine gleitende Mittelwertbildung.

Beispiel 9:

```
# Bilde den Mittelwert über den aktuellen und die letzten 9 Werte des Kanals 1
x1= MEAN1(ch1, 10);
# Bilde den Mittelwert über den aktuellen und die letzten 99 Werte des Kanals 7
x2= MEAN2(ch7, 100);
```

RESET Anweisung

Das RESET Kommando wird im Ausführungsteil einer bedingten Anweisung verwendet. Hiermit können der allgemeine Zähler „CNT“ oder eines der Summenregister auf 0 zurückgesetzt werden.

Form:

IF(<Bedingung>) { RESET(<a>) };

Mit dem Parameter <a> wird angegeben, was zurückgesetzt werden soll:

a=0: Allgemeiner Zähler CNT.

A= 1..23: Summenregister; entsprechend dem Parameter <a> einer verwendeten SUM(<a>) oder MEAN(<a>) Anweisung.

Siehe Beispiel 7:

```
IF(CNT=99) {RESET(0)};
```

CNT nimmt somit Werte von 0 bis 99 an.

Beispiel 10:

```
# Bilde x2= Integral von ch1 in den Grenzen y >10^-6
# Außerhalb dieser Grenzen wird der letzte Wert von x2 wiederholt ausgegeben
x1=ch1/FREQ;
IF(ch1>1E-6){x2=SUM(x1, 0)};
ELIF(ch1<=1E-6){RESET(1)};
ENDIF;
```

Beispiel 10.1:

```
# Wie Beispiel 10, aber x2 wird außerhalb der Integralgrenzen =0 gesetzt:
x1=ch1/FREQ;
IF(ch1>1E-6){x2=SUM(x1, 0)};
```

ELIF(ch1<=1E-6){**x2= 0**};
ENDIF;
IF(ch1<=1E-6){**RESET(1)**};

3. Ausführung des MathScripts

Nachdem das Skript geöffnet und eingebunden wurde (d.h. nach Klicken auf Connect im Add Channel Dialog oder nach Öffnen einer Session mit MathChannels) wird es zunächst ausgewertet („geparst“) und dabei auf einige Fehler geprüft. Viele Fehlertypen können erkannt werden, aber nicht alle. Wird kein Fehler entdeckt, wird das Skript in eine interne Struktur übersetzt, mit deren Hilfe die Berechnungen zur Laufzeit möglichst effizient ausgeführt werden. Dies kann je nach PC-Leistung, Leistungsverfügbarkeit und Skript-Komplexität mit einer Datenfrequenz von bis zu 12000 /s erfolgen.

Alle Laufzeitdaten (z.B. Summenregister und letzte Mess- und Ausgangswerte) werden vor der ersten Ausführung =0 gesetzt. Dieses Reset-Ereignis wird auch beim Klick auf Start Measuring oder Start Recording (im Yt- und XY Recorder) einmalig ausgelöst.

Die Berechnung zur Laufzeit erfolgt in 3 Hauptschritten:

A) Ausführen aller Programmieranweisungen, die keine aktuellen Ausgangswerte (x<no> Variablen) als Input, d.h. rechtsseitig benötigen.

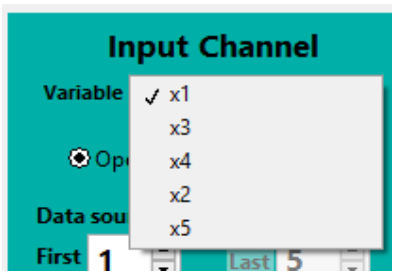
B) Ausführen aller nicht-bedingten arithmetischen Anweisungen

C) Ausführen aller Programmieranweisungen, die aktuelle Ausgangswerte vorangehender Berechnungen (x<no> Variablen) als Input benötigen.

Diese Vorgehensweise bedingt, dass es sein kann, dass die Anweisungen in einer anderen Reihenfolge ausgeführt werden als sie im MathSkript gegeben sind. Außerdem können Ergebnisse von Anweisungen, die in C ausgeführt werden (d.h. die aktuelle Ausgangswerte als Input benötigen), nicht weiter verrechnet werden, d.h. ihre Ergebnisse x<no> dürfen nicht mehr als rechtsseitiger Input-Wert verwendet werden.

Die Ausführungsreihenfolge wird indirekt im Add Channel Dialog angezeigt, nachdem die Skriptdatei geöffnet wurde. Rechts ist unter Input Channel ein Drop-Down Menü namens Variable.

Es ergibt sich z.B. die Reihenfolge x1, x3, x4, x2, x5:



ME-Meßsysteme GmbH
Eduard-Maurer Str. 9
16761 Hennigsdorf
Tel.: +49 3302 78620 60
Fax: +49 3302 78620 69
Mail: info@me-systeme.de
Web: www.me-systeme.de

Bei folgendem Skript (Beispiel 11):

```
x1=MEAN1(ch1,1000);
IF(CNT < 25) {x2= CNT*4*10*x1/100};
ELIF(CNT < 75) {x2= 2*10*x1-(CNT*4*10*x1/100)};
ELIF(CNT < 100) {x2= (CNT*4*10*x1/100)-(4*10*x1)};
ENDIF;
IF(CNT=99) {RESET(0)};

x3= MEAN2(ch2,3);
x4=x1/FREQ;

IF(ch2>1E-6){x5=SUM3(x4,0)};
ELIF(ch2<=1E-6){x5=0};
ENDIF;
IF(ch2<=1E-6){RESET(3)};
```

Dabei wird die Berechnung von x2, d.h. alle Zeilen 2 bis 6 nach Schritt C verschoben, weil diese Anweisungen den Wert x1 rechtsseitig benötigen. x1 (Zeile 1) und x3 (Zeile 7) werden in Schritt A ausgeführt und x4 (Zeile 8) in Schritt B, dabei wird der in A bereits berechnete Wert x1 verwendet.
Die Berechnung von x5 (Zeilen 9 bis 12) wird in Schritt C ausgeführt, weil der Wert x4 rechtsseitig benötigt wird.

Wenn das Skript aus Beispiel 11 in Zeile 7 folgendermaßen geändert wird:

```
x1=MEAN1(ch1,1000);
IF(CNT < 25) {x2= CNT*4*10*x1/100};
ELIF(CNT < 75) {x2= 2*10*x1-(CNT*4*10*x1/100)};
ELIF(CNT < 100) {x2= (CNT*4*10*x1/100)-(4*10*x1)};
ENDIF;
IF(CNT=99) {RESET(0)};

x3= MEAN2( x2, 3);
x4=x1/FREQ;

IF(ch2>1E-6){x5=SUM3(x4,0)};
ELIF(ch2<=1E-6){x5=0};
ENDIF;
IF(ch2<=1E-6){RESET(3)};
```

Dann ist es **nicht** korrekt ausführbar, weil die Berechnung von x3 von einem in Schritt C ausgeführten Berechnungsergebnis, nämlich x2 abhängt.

Die Gesamtanzahl der verwendeten Variablen wird auch durch die Anzahl der Programmieranweisungen im Schritt A zusätzlich eingeschränkt, wobei IF...ELIF...ENDIF Blöcke hier als eine Anweisung gezählt werden. Die Gesamtanzahl der verschiedenen Variablen plus der Anweisungen in A muss kleiner als 287 sein.

3.1.Weitere Tipps zur Verwendung

Wenn man im Add Channel Dialog den Radio Button „Open All Channels“ aktiviert, werden

die MathChannels in der o.g. Ausführungsreihenfolge geöffnet. Man kann Kanäle auch einzeln öffnen, in beliebiger Reihenfolge (wichtig ist dabei nur, dass benötigte Hardware-Kanäle vorher geöffnet wurden).

Um ein anderes MathScript zu verwenden als dasjenige, das bereits eingebunden ist, müssen zunächst alle MathChannels mit Remove Channel entfernt werden.

Wenn Hardware Kanäle für das MathSkript benötigt werden, die nicht angezeigt werden sollen, so kann man diese mit Channel -> Hide "verstecken". Dieser Zustand wird ebenfalls mit Save Session gespeichert.

ME-Meßsysteme GmbH
Eduard-Maurer Str. 9
16761 Hennigsdorf
Tel.: +49 3302 78620 60
Fax: +49 3302 78620 69
Mail: info@me-systeme.de
Web: www.me-systeme.de

Anhang

Liste arithmetischer Funktionen

Function	Name	Description
abs(*x*)	Absolute Value	Returns the absolute value of *x*.
acos(*x*)	Inverse Cosine	Computes the inverse cosine of *x* in radians.
acosh(*x*)	Inverse Hyperbolic Cosine	Computes the inverse hyperbolic cosine of *x*.
asin(*x*)	Inverse Sine	Computes the inverse sine of *x* in radians.
asinh(*x*)	Inverse Hyperbolic Sine	Computes the inverse hyperbolic sine of *x*.
atan(*x*)	Inverse Tangent	Computes the inverse tangent of *x* in radians.
atanh(*x*)	Inverse Hyperbolic Tangent	Computes the inverse hyperbolic tangent of *x*.
ceil(*x*)	Round Toward +Infinity	Rounds *x* to the next higher integer (smallest integer *x*).
cos(*x*)	Cosine	Computes the cosine of *x*, where *x* is in radians.
cosh(*x*)	Hyperbolic Cosine	Computes the hyperbolic cosine of *x*.
cot(*x*)	Cotangent	Computes the cotangent of *x* (1/tan(*x*)), where *x* is in radians.
csc(*x*)	Cosecant	Computes the cosecant of *x* (1/sin(*x*)), where *x* is in radians.
exp(*x*)	Exponential	Computes the value of *e* raised to the *x* power.
expm1(*x*)	Exponential (Arg) – 1	Computes one less than the value of *e* raised to the *x* power ((*e*^*x*) – 1).
floor(*x*)	Round To –Infinity	Truncates *x* to the next lower integer (largest integer *x*).
getexp(*x*)	Get Exponent	Returns the exponent of the input numeric value such that x = mantissa * 2^exponent.
getman(*x*)	Get Mantissa	Returns the mantissa of the input numeric value such that x = mantissa * 2^exponent.
int(*x*)	Round To Nearest	Rounds *x* to the nearest integer.
intrz(*x*)	—	Rounds *x* to the nearest integer between *x* and zero.
ln(*x*)	Natural Logarithm	Computes the natural logarithm of *x* (to the base of *e*).
lnp1(*x*)	Natural Logarithm (Arg +1)	Computes the natural logarithm of (*x* + 1).

log(*x*)	Logarithm Base 10	Computes the logarithm of *x* (to the base of 10).
log2(*x*)	Logarithm Base 2	Computes the logarithm of *x* (to the base of 2).
rand(0)	Random Number (0 – 1)	Produces a floating-point number between 0 and 1 exclusively. Remark: Must write rand(0), i.e. contain dummy argument 0.
sec(*x*)	Secant	Computes the secant of *x*, where *x* is in radians (1/cos(*x*)).
sign(*x*)	Sign	Returns 1 if *x* is greater than 0, returns 0 if *x* is equal to 0, and returns –1 if *x* is less than 0.
sin(*x*)	Sine	Computes the sine of *x*, where *x* is in radians.
sinc(*x*)	Sinc	Computes the sine of *x* divided by *x* (sin(*x*)/*x*), where *x* is in radians.
sinh(*x*)	Hyperbolic Sine	Computes the hyperbolic sine of *x*.
sqrt(*x*)	Square Root	Computes the square root of *x*.
tan(*x*)	Tangent	Computes the tangent of *x*, where *x* is in radians.
tanh(*x*)	Hyperbolic Tangent	Computes the hyperbolic tangent of *x*.

ME-Meßsysteme GmbH
Eduard-Maurer Str. 9
16761 Hennigsdorf
Tel.: +49 3302 78620 60
Fax: +49 3302 78620 69
Mail: info@me-systeme.de
Web: www.me-systeme.de