POSITIONIER- UND BAHNSTEUERUNG APCI-8001 UND APCI-8008

PROGRAMMIER- UND REFERENZ-HANDBUCH / PHB (TEIL 2)

Stand: 31.03.2021, ab Disk V2.53VP
Rev. 18/052022

www.addi-data.de

5 Die Programmiersprache rw_SymPas für die Standalone-Applikations-Programmierung

5.1 Einführung

rw_SymPas ist eine Programmiersprache zur Erzeugung von selbständig ablauffähigen CNC-Programmen (Stand-Alone-Applikations-Programmen) für die Positioniersteuerung APCI-800x. Die lexikalische und semantische Grammatik von *rw_SymPas* ist stark an die Programmiersprache *Pascal* angelehnt.

5.2 Lexikalische Grammatik

Dieses Kapitel enthält eine formale Definition der lexikalischen Grammatik von rw_SymPas. Diese befasst sich mit den wortähnlichen Einheiten einer Sprache, sogenannten »Symbolen« oder »Tokens«. Die semantische Grammatik bestimmt die Regeln, nach denen Symbole zur Bildung von Ausdrücken, Anweisungen oder anderen Einheiten kombiniert werden können.
In *rw_SymPas* ergeben sich die Symbole als Folge der Operationen, die der *NCC*-Compiler mit dem Benutzerprogramm durchführt. Ein *rw_SymPas* Programm ist eine Abfolge von ASCII-Zeichen, die den Quellcode darstellen, der mit einem Texteditor (z.B. CNC-Edit) erstellt wurde. Die grundlegende Programmeinheit in *rw_SymPas* ist die Datei. Sie entspricht einer benannten DOS-Datei im Speicher oder auf der Platte und hat die Namenserweiterung .SRC.

5.2.1 Whitespace

In der lexikalischen Analysephase der Compilierung wird die Quellcodedatei in Symbole und »Whitespace« geparst (d.h. zerlegt). Whitespace ist der Sammelbegriff für Zeichen, die als Trenner gelten: Leerzeichen, Tabulatoren, Zeilenvorschübe und Kommentare. Whitespace dient zur Markierung von Beginn und Ende eines Symbols, aber abgesehen davon wird Whitespace ignoriert.

5.2.2 Kommentare

Kommentare sind Textzeilen, die Erklärungen zum Programm enthalten. Sie werden vor dem Parsen aus dem Quelltext entfernt.
Ein *rw_SymPas* Kommentar ist eine Zeichenfolge, die nach dem Zeichen { steht. Der Kommentar endet bei dem ersten Auffinden des Zeichens }, welches auf das Startsymbol { folgt. Die Verschachtelung von Kommentaren ist nicht zulässig.
Weiterhin ist es möglich einen einzeiligen Kommentar mit zwei Schrägstrichen // anzulegen. Der Kommentar kann an beliebiger Stelle beginnen und erstreckt sich bis zur nächsten Zeile.

5.2.3 Symbole

rw_SymPas kennt folgende Arten von Symbolen

Symbol:
Schlüsselwort
Bezeichner
Qualifizierte Bezeichner
Labels
Konstante
Operator
Interpunktionszeichen (auch Trennzeichen)

5.2.3.1 Schlüsselwörter

Schlüsselwörter sind für spezielle Zwecke reservierte Wörter, die nicht als normale Bezeichnernamen verwendet werden dürfen. In der folgenden Tabelle sind alle *rw_SymPas* Schlüsselwörter aufgelistet.

Tabelle 21: Alle *rw_SymPas* Schlüsselwörter

and	begin	boolean	const
do	double	downto	else
end	for	forward	function
goto	if	integer	label
mod	module	not	or
procedure	repeat	shl	shr
single	then	timer	to
until	var	while	xor

5.2.3.2 Bezeichner

Bezeichner können aus folgenden Elementen bestehen:

Bezeichner
Nicht-Ziffer
Bezeichner Nicht-Ziffer
Bezeichner Ziffer

Nicht-Ziffer: eines der folgenden Zeichen
a b c d e f g h i j k l m n o p q r s t u v w x y z _
A B C D E F G H I J K L M N O P Q R S T U V W X Y Z

Ziffer: eines der folgenden Zeichen
0 1 2 3 4 5 6 7 8 9

Beispiele:

```
A, AA, AB, A1, A2, _A          // gültig
1A, ?B                         // ungültig
```

5.2.3.2.1 Namen- und Längenbeschränkung

Bezeichner sind beliebige Namen von einer beliebigen Länge für Variablen, Prozeduren, Funktionen, Labelnamen, etc. Bezeichner können die Buchstaben A bis Z, a bis z, den Unterstrich und die Ziffern 0 bis 9 enthalten. Es gibt jedoch folgende Einschränkungen:

- Das erste Zeichen muss ein Buchstabe oder ein Unterstrich sein.
- Nur die ersten 32 Zeichen sind signifikant. Sofern der Bezeichner mehr als 32 Zeichen enthält, werden die restlichen Zeichen verworfen. Bei grossen *rw_SymPas* Programmen sollte man sich auf kurze Namen beschränken, um den Arbeitsspeicher des PC zu entlasten.
- In DIN G-Code-Programmen sind keine numerischen Zeichen erlaubt.

5.2.3.2.2 Bezeichner Groß- und Kleinschreibung

In *rw_SymPas* wird zwischen Groß- und Kleinschreibung unterschieden, so dass *Position*, *position* und *positioN* unterschiedliche Bezeichner sind.

5.2.3.2.3 Eindeutigkeit und Gültigkeit von Bezeichnern

Bezeichner können beliebige Namen sein, die den geltenden Regeln entsprechen. Es kann jedoch zu Fehlern kommen, wenn derselbe Name innerhalb desselben Geltungsbereichs für mehrere Bezeichner verwendet wird, die denselben Namensbereich haben. Gleiche Namen sind für verschiedene Namensbereiche zulässig, unabhängig vom Geltungsbereich. Die Definition des Geltungsbereichs von Bezeichnern wird im Kapitel 5.3.2.2 erläutert.

5.2.3.3 Standardbezeichner

rw_SymPas definiert bereits eine Reihe von Bezeichnern, für die deshalb der Name Standardbezeichner verwendet wird. In der folgenden Tabelle sind alle *rw_SymPas* Standardbezeichner aufgelistet.

Tabelle 22: Alle *rw_SymPas* vordefinierten Standardbezeichner

abort	cl	contcnct	disev
enev	ja	jaw	jhi
jhiw	jhl	jhlw	jhr
jhrw	jr	jrw	js
jsw	mca	mcaw	mcr
mcrw	mha	mhaw	mhr
mhrw	mla	mlaw	mlr
mlrw	ms	msw	ol
ra	rs	shp	smca
smcr	smha	smhr	smla
smlr	ssms	ssmsw	startcnct
stop	stepcnct	stopcnct	uf
wt	utrovr		

5.2.3.4 Achsenbezeichner

Jeder Achskanal wird mit Hilfe eines symbolischen Namens referenziert. Dieser Name kann mit bis zu 8 Zeichen vom Benutzer frei gewählt werden. Diese Achsenbezeichner werden von *rw_SymPas* ebenfalls in die Standardbezeichnerliste mitaufgenommen.
Anmerkung: Die automatische Deklaration der Achsenbezeichner weicht vom Standard-Pascal ab.

5.2.3.5 Qualifizierte Bezeichner

Der Bezug auf Bezeichner gleichen Namens, die für verschiedene Achssysteme (von *rw_SymPas*) deklariert sind, geschieht über eine Qualifizierung durch Voranstellen des Achsenbezeichners.
Beispiele:

```
A1.digo := 0;                          // alle Ausgänge der APCI-800x rücksetzen
A2.digo := $FFFFFFFF;     // alle Ausgänge der APCI-800x setzen
```

Anmerkung: Der Variablenbezug auf qualifizierte Bezeichner weicht vom Standard-Pascal ab.

5.2.3.6 Labels

Für den Aufbau eines Labels gelten dieselben Regeln wie für die Bezeichner. Labels werden ausschließlich im Zusammenhang mit der Anweisung *goto* verwendet.

5.2.3.7 Konstanten

Konstanten sind Symbole, die für feste numerische Werte stehen. *rw_SymPas* kennt zwei Klassen von Konstanten: Gleitkomma und Integer. Der Datentyp einer Konstante wird vom *NCC*-Compiler auf Grund ihres numerischen Wertes und ihres Formats im Quelltext abgeleitet. Tabelle 23 zeigt die formale Definition einer Konstante.

Tabelle 23: Formale Definition einer Konstanten.

Konstante: Gleitpunktkonstante Integerkonstante
Gleitpunktkonstante: Fraktionale-Konstante<Exponent> Ziffernfolge Exponent
Fraktionale Konstante: <Ziffernfolge>.Ziffernfolge Ziffernfolge.
Exponent: e<Vorzeichen>Ziffernfolge E<Vorzeichen>Ziffernfolge
Vorzeichen: Eines der folgenden Zeichen + -
Ziffernfolge: Ziffer Ziffernfolge Ziffer
Integer-Konstante: <Vorzeichen>Dezimale Konstante Hexadezimale Konstante
Dezimale-Konstante: Ziffer Dezimale-Konstante Ziffer
Hexadezimale-Konstante: $ Hexziffer Hexadezimale-Konstante Hexziffer
Ziffer: 0 1 2 3 4 5 6 7 8 9

Hexziffer:
0 1 2 3 4 5 6 7 8 9
a b c d e f
A B C D E F

5.2.3.7.1 Integer-Konstanten

Integerkonstanten können dezimale (Basis 10) oder hexadezimale (Basis 16) Zahlen sein. Zu beachten ist, dass für dezimale und nichtdezimale Konstanten unterschiedliche Regeln gelten.

5.2.3.7.1.1 Dezimalkonstanten

Es sind Dezimalkonstanten von -2147483648 bis 2147483647 zugelassen. Konstanten außerhalb des Bereichs werden automatisch auf den entsprechenden Minimal- bzw. Maximalwert begrenzt.

5.2.3.7.1.2 Hexadezimale Konstanten

Alle Konstanten, die mit dem Dollarzeichen ($) beginnen, werden als hexadezimale Konstante interpretiert. Hexadezimale Konstanten von $80000000 bis $7FFFFFFF sind zugelassen. Konstanten außerhalb des Bereichs werden auf den entsprechenden Minimal- bzw. Maximalwert begrenzt.

5.2.3.7.2 Gleitpunkt-Konstanten

Eine Gleitpunktkonstante setzt sich aus 4 Bestandteilen zusammen:

- Vorkommastellen
- Dezimalpunkt
- Nachkommastellen
- e oder E und ein vorzeichenbehafteter Integer-Exponent (optional)

Es können entweder die Vorkomma- oder Nachkommastellen wegfallen (aber nicht beide). Der Dezimalpunkt oder der Buchstabe e (E) kann weggelassen werden (aber nicht beide). Diese Regeln ermöglichen sowohl die konventionelle als auch die wissenschaftliche Notation (mit Exponenten).

5.2.3.7.2.1 Der Typ der Gleitkommakonstanten

Gleitkommakonstanten werden grundsätzlich als Double-Werte behandelt. Sie werden in einem Doppelwort (8 Byte) nach IEE abgelegt. Der Bereich ist $1.7*10^{-308}$ bis $1.7*10^{308}$.

5.2.3.7.2.2 Deklaration von Konstanten

Eine Konstanten-Deklaration vereinbart einen Bezeichner, der innerhalb des entsprechenden Blocks für einen konstanten Wert steht. Beispiel Konstantendeklaration:

```
const   one = 1;
```

Vorzeichenbehaftete Konstanten stehen für einen Ganzzahl- oder Gleitkommawert. Die Berechnung von Konstanten ist nicht möglich.

5.2.3.7.3 Interpunktionszeichen

Interpunktionszeichen (auch Trennzeichen genannt) sind in *rw_SymPas* wie folgt definiert:

Interpunktionszeichen: eines der folgenden Symbole
() , ; : =

5.2.3.7.3.1 Runde Klammern

() fassen Ausdrücke zusammen, isolieren konditionale Ausdrücke und repräsentieren Prozeduraufrufe und Prozedurparameter:

```
d := c * (a + b);          // Ändern der normalen Reihenfolge
if (d = z) then ...        // Erforderlich bei einer bedingten
                                  // Anweisung
proc()                     // Prozeduraufruf ohne Argumente
```

5.2.3.7.3.2 Komma

Das Komma (,) trennt die Elemente in einer Prozedur-Argumentliste:

```
mlr (A1, A2);
```

5.2.3.7.3.3 Strichpunkt

Der Strichpunkt (;) dient als Endekriterium einer Anweisung. Jeder gültige *rw_SymPas* Ausdruck (auch der leere Ausdruck), an dessen Ende ein Strichpunkt steht, wird als Anweisung (Ausdruck-Anweisung) interpretiert.

5.2.3.7.3.4 Gleichheitszeichen

Das Gleichheitszeichen (=) trennt Konstantendeklarationen von den Initialisierungswerten:

```
Const one = 1.0;
```

5.3 Semantische Grammatik

In diesem Kapitel wird die formale Definition von der *rw_SymPas* Sprachstruktur erläutert. Diese semantische Grammatik bestimmt die Regeln, nach denen Symbole zur Bildung von Ausdrücken, Anweisungen oder anderen bedeutungstragenden Einheiten kombiniert werden können.

5.3.1 Deklarationen

Im folgenden Abschnitt befindet sich eine kurze Zusammenfassung über Themen, die mit Deklarationen zu tun haben: Objekte, Typen, Blöcke, Lokalität und Geltungsbereich. Lokalität und Geltungsbereich legen diejenigen Programmteile fest, von denen aus ein zulässiger Zugriff auf das mit dem Bezeichner verknüpfte Objekt möglich ist.

5.3.1.1 Objekte

Ein Objekt ist ein identifizierbarer Speicherbereich, in dem ein fester oder variabler Wert (oder eine Menge von Werten) steht. Jedes Objekt hat einen Namen und einen Typ (den sogenannten Datentyp). Der Zugriff auf das Objekt erfolgt über seinen Namen. Dieser Name kann ein einfacher Bezeichner oder ein komplexer Ausdruck sein, der eindeutig auf ein Objekt zeigt. Der Typ wird dazu verwendet um:

- die korrekte Speicherreservierung festzulegen, die zu Beginn erforderlich ist
- durch Überprüfung der Typen sicherzustellen, dass zulässige Zuweisungen erfolgen

Zu den vordefinierten Typen von *rw_SymPas* gehören der Datentyp Boolean, Integerzahlen mit Vorzeichen und Gleitkommazahlen mit unterschiedlicher Genauigkeit.
Deklarationen stellen die Verbindung zwischen Bezeichnern und Objekten her. Jede Deklaration verknüpft einen Bezeichner mit einem Datentyp. Darüber hinaus bestimmen die meisten Deklarationen, die sogenannten Definitionsdeklarationen, auch die Generierung des Objekts (wo und wann) und übernehmen die Zuweisung des Speicherplatzes.

5.3.1.2 Typen

Jede Deklaration einer Variablen muss den Typ dieser Variablen angeben. Der Typ legt den Wertebereich der Variablen fest und bestimmt die Operationen, die mit ihr ausgeführt werden können. Eine Typendefinition vereinbart also einen Bezeichner, der seinerseits für einen bestimmten Typ steht.

Typen-Deklaration:
    Bezeichner = Typ;

Typ:
    Boolean-Typ
    Ganzzahl-Typ
    Gleitpunkt-Typ

5.3.1.2.1 Boolean-Typ

Der Datentyp *Boolean* kann ausschließlich einen der vordefinierten Werte *FALSE* oder *TRUE* annehmen. Dabei gelten folgende Beziehungen:

- *FALSE* < *TRUE*
- Ordinalzahl von *FALSE* = 0
- Ordinalzahl von *TRUE* = 1

5.3.1.2.2 Ganzzahl-Typ

rw_SymPas stellt die Ganzzahl-Typen *Integer* und *Timer* zur Verfügung.

Tabelle 24: Der Ganzzahl-Typ und sein Wertebereich

Typ	Bereich	Format
Integer	-2147483648 .. 2147483647	32 Bit mit Vorzeichen
Timer	0 .. 4294967295	32 Bit ohne Vorzeichen

5.3.1.2.3 Gleitpunkt-Typen (Real-Typen)

rw_SymPas kennt zwei verschiedene Arten von Gleitpunkt-Typen: *Single* und *Double*. Die Typen unterscheiden sich voneinander sowohl durch ihren Wertebereich als auch in der Genauigkeit mit ihnen durchgeführter Operationen.
Anmerkung: Gelegentlich wird auch der Begriff Real-Typ für Gleitpunkt-Typ verwendet.

Tabelle 25: Die Gleitpunkt-Typen und ihre Genauigkeit

Typ	Bereich	Format
Single	$-1.2e^{-38}$.. $3.4e^{38}$	7 bis 8 Stellen
Double	$-2.2e^{-308}$ .. $1.8e^{308}$	15 bis 16 Stellen

5.3.1.2.4 Zuweisungskompatibilität von Typen

Die Zuweisungskompatibilität ist unabdingbar, wenn ein Wert zugewiesen werden soll. Der Wert eines Typs T_2 kann einem Wert T_1 zugewiesen werden (d.h. $T_1:=T_2$), wenn eine der folgenden Bedingungen erfüllt ist:

- T_1 und T_2 sind vom gleichen Typ.
- T_1 hat den Typ Double, T_2 den Wert Integer oder Single.
- T_1 hat den Typ Single, T_2 den Wert Integer.

Wenn keine dieser Bedingungen erfüllt, Zuweisungskompatibilität aber erforderlich ist, meldet der *NCC*-Compiler einen Fehler.

5.3.1.3 Variablen

5.3.1.3.1 Automatische Typ-Konvertierung

rw_SymPas führt eine automatische Typ-Konvertierung durch, sofern mit unterschiedlichen Typen in einem Ausdruck operiert wird. Die Konvertierung wird wie folgt ausgeführt: Integer nach Single oder Integer und Single nach Double. Beispiel:

```
...
Var
        i : Integer;
        s : Single;
        d : Double;
...

d := s * i;                    // s und i werden automatisch nach Double konvertiert

s := i;   // i wird automatisch nach Single konvertiert
```

5.3.2 Blöcke, Lokalität und Geltungsbereich

Ein Block besteht aus beliebig angeordneten Deklarationen und Anweisungen. Jeder Block ist Teil einer Prozedur-Deklaration oder eines Programms. Alle Bezeichner und Labels im Deklarationsteil des Blocks sind in ihrer Wirkung auf diesen Block beschränkt - sie sind lokal zu diesem Block.

5.3.2.1 Syntax

Der syntaktische Aufbau jedes Blocks lässt sich wie folgt darstellen:

Block:
    Deklarationsteil
    Befehlsteil

5.3.2.1.1 Deklarationsteil

Deklarationsteil:
    Label-Deklarationsteil
    Konstanten-Deklarationsteil
    Variablen-Deklarationsteil
    Deklarationsteil Label-Deklarationsteil
    Deklarationsteil Konstanten-Deklarationsteil
    Deklarationsteil Variablen-Deklarationsteil

5.3.2.1.1.1 Label-Deklarationsteil

Im *Label-Deklarationsteil* werden alle Labels vereinbart, die *goto*-Sprungziele im Befehlsteil des betreffenden Blocks darstellen sollen. Jedes Label darf innerhalb des Befehlsteils nur einmal definiert werden (d.h. jedes *goto* muss ein eindeutiges Ziel haben).

Aufbau des Label-Deklarationsteils:

label Labels;

Labels:
LabelName
Labels, LabelName

5.3.2.1.1.2 Konstanten-Deklarationsteil

Der Deklarationsteil für Konstanten beinhaltet alle Vereinbarungen von Konstanten, die lokal zum entsprechenden Block sind.

Aufbau des Konstanten-Deklarationsteils:

const Konstanten-Deklarationen

Konstanten-Deklarationen:
Konstanten-Deklaration
Konstanten-Deklarationen Konstanten-Deklaration

5.3.2.1.1.3 Variablen-Deklarationsteil

Der Deklarationsteil für Variablen beinhaltet alle Variablen-Deklarationen, die lokal zum entsprechenden Block sind.

Aufbau des Variablen-Deklarationsteils:

var Variablen-Deklarationen

Variablen-Deklarationen:
Variablen-Deklaration
Variablen-Deklarationen Variablen-Deklaration

5.3.2.1.2 Befehlsteil

Im Befehlsteil werden alle Operationen definiert, die bei der Aktivierung des Blocks ausgeführt werden.

Befehlsteil:
Verbundanweisung

Zulässige Verbundanweisungen werden im Kapitel 5.3.5.5 erläutert.

Der Befehlsteil des Hauptprogrammblocks hat folgenden Aufbau:

begin
Anweisungsfolge;
end.

5.3.2.2 Geltungsbereich

Jeder Bezeichner und jedes Label einer Deklaration vereinbart genau ein Objekt bzw. ein Sprungziel. Deshalb muss ein Bezeichner, genauso wie ein Label, immer im Geltungsbereich seiner Deklaration sein, wenn er im Programm erscheint. Der Geltungsbereich von Bezeichnern und Labels liegt zwischen der eigentlichen Deklaration und dem Ende des dazugehörigen Blocks, wobei alle Blöcke mit eingeschlossen sind, die dieser Block umfasst. Allerdings gibt es hierzu einige Ausnahmen, die in den folgenden Absätzen erläutert werden.

5.3.2.2.1 Neudeklaration in einem untergeordneten Block

Mit der Annahme, dass ein Block »Aussen« einen Block »Innen« umfasst, d.h. ihm übergeordnet ist, beschränkt jede Neudeklaration eines Bezeichners von »Aussen« im Block »Innen« den Geltungsbereich dieses Bezeichners auf den Block »Innen«. Anders formuliert: Wenn »Aussen« eine Variable *x* deklariert und »Innen« eine Variable gleichen Namens, dann können Anweisungen im Block »Innen« nicht auf die in »Aussen« deklarierte Variable *x* zugreifen.

5.3.2.2.2 Der Ort einer Deklaration im Block

Bezeichner und Labels müssen deklariert sein, bevor sie in einem Block benutzt werden können. Auf Zugriffsversuche vor ihrer eigentlichen Deklaration reagiert der *NCC*-Compiler mit der Fehlernummer 3.

5.3.2.2.3 Neudeklarationen innerhalb eines Blocks

Bezeichner und Labels können auf der obersten Ebene eines Blocks nur jeweils einmal deklariert werden, es sei denn, ihre Neudeklaration erfolgt innerhalb eines untergeordneten Blocks.

5.3.2.2.4 Standardbezeichner

rw_SymPas bietet eine ganze Reihe vordefinierter Konstanten, Typen und Prozeduren, die so arbeiten, als wären sie innerhalb eines Blocks deklariert worden, der das ganze Programm umschließt. Folglich umfasst ihr Geltungsbereich das gesamte Programm.

5.3.3 Variablen

5.3.3.1 Die Deklaration von Variablen

Die Variablen-Deklaration enthält eine Liste von Bezeichnern, die ihrerseits für neue Variablen und ihren jeweiligen Typ stehen.

Variablendeklaration:
Bezeichnerliste: Typ;

Bezeichnerliste:
Variablenname
Variablennamen, Variablenname

Typ:
BOOLEAN
INTEGER
SINGLE
TIMER
DOUBLE

Beispiele für gültige Variablendeklarationen sind:

```
var
    on, off:    BOOLEAN;
    eins:       INTEGER;
    dvalue:     DOUBLE;
    ticks:      TIMER;
```

Wenn ein Bezeichner in der Bezeichnerliste eines Deklarationsteils steht, gilt er innerhalb des gesamten Blocks, für den er deklariert wurde. Es kann über den ganzen Block hinweg auf diese Variable Bezug genommen werden, solange nicht in einem untergeordneten Block derselbe Bezeichner für eine andere Variable verwendet wird (»Redeklaration«). Eine redeklarierte Variable benutzt den Namen eines bereits existierenden Bezeichners, stellt aber ansonsten eine eigenständige Einheit dar. Der Wert der ursprünglichen Variablen wird durch die Redeklaration nicht beeinflusst. Variablen die außerhalb von Prozeduren oder Funktionen deklariert sind, werden als *global* bezeichnet. Innerhalb von Prozeduren oder Funktionen deklarierte Variablen sind *lokal*.

5.3.3.1.1 Axis-Typ-Deklaration

Mit Hilfe der AXIS-Typ Deklaration können variable Achsen definiert werden. Dies ist insbesondere für die Gestaltung von mehrfach verwendeten Unterprogrammen (Prozeduren/Funktionen) hilfreich in denen immer wiederkehrende Aktionen für verschiedene Achsen ausgeführt werden sollen.

Beispiel:

```
var
    VA : AXIS;                      // variable Achse mit Namen VA
    VA.an := 0;                     // Achsennummer 0 an VA zuweisen (wichtig!)
    ol(VA);                 // open loop von Achse 0
    for VA.an := 0 to 5 do  cl(VA);         // close loop Achse 0 .. 5
```

Anmerkung: Auch die Achsnummer der vordefinierten Achsenspezifizierer (A1 .. An), welche bei den Systemparametern definiert sind, können auf diese Weise neu vergeben werden. Dies kann jedoch schnell zu einer unübersichtlichen und fehlerhaften Programmierung führen. Sobald mit variablen Achsen gearbeitet werden soll, empfiehlt es sich entsprechende Variablen mit entsprechendem Symbolcharakter zu verwenden.

Achtung: Momentan darf die Deklaration von variablen Achsen nur im Hauptprogramm erfolgen, also nicht in Prozeduren oder Funktionen!

5.3.3.1.2 Timer-Deklaration

Eine Eingabe mit Hilfe der vordefinierten System-Variablen *CLOCK* liefert einen Wert vom Typ *Timer*, welcher die Zeit darstellt. Dieser Wert wird von einer internen Uhr der Steuerung geliefert, die in regelmäßigen Zeitabständen ihren Wert ändert. Dieser Wert läuft zyklisch weiter, d.h. nach dem größten positiven Wert wird als nächstes der kleinste negative Wert geliefert. Das Zeitintervall, in der diese interne Uhr hochgezählt wird, beträgt 64µs.
Mit Hilfe *von CLOCK* kann jederzeit der Zählerstand dieser Uhr einer *Integer* oder *Timer*-Variablen zugewiesen werden. Sofern verschiedene Zeiten miteinander zu vergleichen sind, sollte dieser Vergleich nur mit Hilfe von *Timer*-Variablen durchgeführt werden, da hier ein Timer-Überlauf beim Vergleichsoperator > automatisch berücksichtigt wird. Ebenso wird die Addition und Subtraktion mit *Timer*-Variablen in Modulo-Technik, also vorzeichenlos durchgeführt. Hingegen kann bei *Integer*-Variablen ein Über- bzw. Unterlauf stattfinden, welcher wiederum das Setzen eines Internen Error-Flags nach sich zieht und in bestimmten Situationen zum Abbruch des *rw_MOS*-Betriebssystems führt.
Eine praktische Timer-Anwendung könnte etwa so aussehen:

```
Const
        s := 15625;                     // 15625 ticks = 1s
Var
        t: timer

t := CLOCK + 5*s;          // Verzögerung von 5s ab jetzt berechnen
...
repeat
        ...
until CLOCK > t;                        // warten bis 5s vorbei
...
```

In diesem Beispiel wird ersichtlich, dass die Addition von *CLOCK* und der Verzögerungszeit bei großen Werten von *CLOCK* zum Überlauf führt. Deshalb empfiehlt es sich für die Deklaration der Variable *t* den *Timer* anstatt den *Integer*-Typ zu verwenden. Ein weiterer Grund ist die Abfrage auf das Erreichen der berechneten Verzögerungszeit. Im Fall eines Überlaufs bei der Berechnung der Verzögerungszeit ist der Inhalt von *t* nämlich kleiner als *CLOCK*. Dieser Umstand wird durch die Deklaration als *Timer*-Variable ebenfalls richtig behandelt.
Der Wertebereich einer Timer-Variable liegt zwischen 0 und 4294967295. Es können Verzögerungszeiten bis zu 38 h realisiert werden.

5.3.3.2 Umwandlung von Variablentypen

Der Bezug auf eine Variable eines bestimmten Typs kann in den Bezug auf eine Variable eines anderen Typs umgewandelt werden.

Typ Umwandlung:
Typbezeichner (Variablenbezug)

Typbezeichner:
BOOLEAN
INTEGER
SINGLE
DOUBLE

Einige Beispiele für die Umwandlung von Variablen-Typen:

```
var
        B : BOOLEAN;
        I : INTEGER;
        D : DOUBLE;

        B := BOOLEAN (I);
        B := BOOLEAN (D);
        D := B;
        I := INTEGER (D);
        I := B;
```

5.3.4 Ausdrücke

Ausdrücke bestehen aus Operatoren und Operanden. Die meisten Operatoren von *rw_SymPas* verknüpfen zwei Operanden und werden deshalb als binär bezeichnet. Die restlichen Operatoren arbeiten mit nur einem Operanden, daher bezeichnet man sie als unär. Binäre Operatoren benutzen die herkömmliche algebraische Form wie z.B. *a+b*. Ein unärer Operator steht immer unmittelbar vor seinem Operanden, wie bei *-b*. Bei umfangreichen Ausdrücken regelt die in Tabelle 26 gezeigte *Rangfolge* der Operatoren die Reihenfolge der Berechnung. Es gelten drei grundlegende Regeln:

- Ein Operand zwischen zwei Operatoren von unterschiedlichem Rang ist immer an den höherrangigen Operator gebunden.
- Ein Operand zwischen gleichrangigen Operatoren ist immer an den Operator gebunden, der links von ihm steht.
- Ausdrücke in Klammern werden als einzelner Operand betrachtet und immer als erstes ausgewertet.

Tabelle 26: Rangfolge der Operatoren

Operatoren	Rangfolge	Kategorie
-, +, not	1 (am höchsten)	unär
*, /, mod, shl, shr, and	2	multiplizierend
+, -, or, xor	3	addierend
=, <>, <, >, <=, >=	4	relational

Operationen von gleichem Rang werden normalerweise von links nach rechts durchgeführt.

5.3.4.1 Syntax von Ausdrücken

Die Rangfolge der Operatoren folgt der Syntax von Ausdrücken, die sich aus Faktoren, Termen und einfachen Ausdrücken zusammensetzen. Faktoren lassen sich durch folgende Syntax darstellen:

Faktor:
- Variablenbezug
- vorzeichenlose Konstante
- (Ausdruck)
- *not* Faktor
- Typ-Umwandlung (Werte)

vorzeichenlose Konstante:
- vorzeichenloser numerischer Wert
- Zeichenfolge
- Konstanten-Bezeichner

Die folgenden Angaben stellen gültige Faktoren dar:

Dummy	Variablenbezug
15	vorzeichenlose Konstante

5.3.4.2 Operatoren

Operatoren werden in vier Gruppen unterschieden: Arithmetische, logische, boolsche und relationale Operatoren.

5.3.4.3 Arithmetische Operatoren

Folgende Tabellen zeigen die Operanden- und Ergebnistypen binärer bzw. unärer arithmetischer Operationen.

Tabelle 27: Binäre arithmetische Operatoren

Operator	Operation	Operandentyp	Ergebnistyp
+	Addition	Integer, Real	Integer, Real
-	Subtraktion	Integer, Real	Integer, Real
*	Multiplikation	Integer, Real	Integer, Real
/	Division	Integer, Real	Integer, Real
mod	Modulo	Integer	Integer

Anmerkung: Sofern einer der Operanden den Typ *Timer* hat, werden Addition und Subtraktion mit Hilfe der Modulo-Technik durchgeführt. Es erfolgt keine Überlaufprüfung, da die *Timer*-Werte zyklisch sind. Weitere Angaben zum *Timer*-Typ sind im Kapitel 5.3.3.1.2 enthalten.

Tabelle 28: Unäre arithmetische Operatoren

Operator	Operation	Operandentyp	Ergebnistyp
+	Identität	Integer, Real	Integer, Real
-	Negation	Integer, Real	Integer, Real

Wenn beide Operanden eines Operators +, -, *, /, oder *mod* einen Integer-Typ haben, hat das Ergebnis ebenfalls den Typ *Integer*. Ist einer der Operanden eines Operators +, -, * oder / vom Typ *Real*, dann hat das Ergebnis ebenfalls den Typ *Real*.
Der Operator *mod* liefert den Rest der Division seiner Operanden zurück, also:

```
i mod j = i - (i/j)*j;
```

5.3.4.4 Logische Operatoren

Tabelle 29 gibt die Typen der Operanden und Ergebnisse logischer Operationen wieder.

Tabelle 29: Logische Operationen

Operator	Operation	Operandentyp	Ergebnistyp
not	bitweise Negation	Integer	Integer
and	bitweises UND	Integer	Integer
or	bitweises ODER	Integer	Integer
xor	bitweise Antivalenz	Integer	Integer
shl	Linksschieben	Integer	Integer
shr	Rechtsschieben	Integer	Integer

Anmerkung: *not* ist ein unärer Operator.

Die Operationen i *shl* j und i *shr* j verschieben den Wert von *i* um *j* Bitpositionen nach links bzw. nach rechts, entsprechen also einer Multiplikation bzw. Division mit 2^j.

5.3.4.5 Boolsche Operatoren

Tabelle 30 gibt die Typen der Operanden und Ergebnisse boolscher Operationen wieder.

Tabelle 30: Boolsche Operatoren

Operator	Operation	Operandentyp	Ergebnistyp
not	logische Negation	Boolean	Boolean
and	logisches UND	Boolean	Boolean
or	logisches ODER	Boolean	Boolean
xor	logische Antivalenz	Boolean	Boolean

Anmerkung: Der Operator **not** ist auch hier unär.

Bei Operanden des Typs *Boolean* bestimmt die normale boolsche Logik das Ergebnis dieser Operationen. Beispielsweise ergibt *a and b* nur dann *TRUE* (wahr), wenn *a* und *b* wahr sind.

5.3.4.6 Relationale Operatoren

Tabelle 31 gibt die Operandentypen und die Ergebnisse relationaler Operationen wieder.

Tabelle 31: Relationale Operatoren

Operator	Operation	Operandentyp	Ergebnistyp
=	gleich	Integer, Real	Boolean
<>	ungleich	Integer, Real	Boolean
<	kleiner als	Integer, Real	Boolean
>	größer als	Integer, Real	Boolean
<=	kleiner gleich	Integer, Real	Boolean
>=	größer gleich	Integer, Real	Boolean

Anmerkung: Sofern einer der Operanden den Typ *Timer* hat wird die Operation *größer als* (>) mit Hilfe der Modulo-Technik durchgeführt. Es erfolgt keine Überlaufprüfung, da die *Timer*-Werte zyklisch sind. Weitere Angaben zum *Timer*-Typ sind im Kapitel 5.3.3.1.2 enthalten.

5.3.5 Anweisungen

Der engl. Begriff *statement* steht für alle Konstrukte, die eine durch die APCI-800x ausführbare Aktion vereinbaren. In diesem Hanbuch wird der Terminus »Anweisung« als Oberbegriff für Anweisungen (wie *begin, end* oder *for*) und *Befehle* (wie *goto*, Zuweisungen, Prozedur-Aufrufe usw.) verwendet.
Jeder Anweisung (d.h. jeder Vereinbarung einer ausführbaren Aktion) kann ein Label vorangestellt werden, auf das wiederum ein Bezug mit *goto* möglich ist: ein *goto* zu diesem Label bewirkt einen direkten Sprung zu dieser Anweisung und ihre Ausführung.

Aufbau einer Anweisung:

Label: Anweisung

Anweisung:
Zuweisung
Prozeduranweisung
goto-Anweisung

5.3.5.1 Zuweisungen

Zuweisungen ersetzen den momentanen Wert einer Variablen mit einem neuen Wert, der über einen Ausdruck angegeben wird.

Aufbau einer Zuweisung:

Variablenbezug **:=** Ausdruck

5.3.5.2 Prozedur- oder Funktionsaufrufe

Durch Angabe eines Prozedurbezeichners wird eine Prozedur aufgerufen, die mit diesem Bezeichner deklariert wurde. Die Übergabe von Parametern an die Prozedur wird erst ab mcfg.exe V2.5.3.97 (ncc.exe/dll V2.5.3.73) unterstützt. Durch Angabe eines Funktionsbezeichners wird eine Funktion aufgerufen, die mit diesem Bezeichner deklariert wurde. Die Verwendung von anwenderdefinierten Funktionen wird erst ab mcfg.exe V2.5.3.97 (ncc.exe/dll V2.5.3.73) und RWMOS.ELF ab V2.5.3.126 unterstützt.

5.3.5.3 Die goto-Anweisung

führt einen Sprung zum angegebenen Label aus: Das Programm wird an dem Punkt fortgesetzt, der unmittelbar auf das Label folgt. Die Syntax von *goto* ist:

goto Label

Bei Verwendung von *goto* müssen folgende Regeln beachtet werden:

- Das Label, auf das *goto* Bezug nimmt, muss im selben Block stehen wie die *goto*-Anweisung selber. Es ist nicht möglich, mit *goto* beliebig zwischen Prozeduren/Funktionen hin- und herzuspringen.
- Der Bezug auf einen strukturierten Anweisungsblock von einem Programmteil außerhalb dieses Blocks (d.h ein Sprung auf eine tiefere Verschachtelungsebene) kann unvorhersehbare Folgen haben. *rw_SymPas* kann derartige Fehler nicht erkennen.

5.3.5.4 Strukturierte Anweisungen

bestehen aus mehreren ineinander verschachtelten Ebenen, die ihrerseits Anweisungen enthalten. Sie werden entweder in der Reihenfolge ihres Erscheinens (Verbund-Anweisungen), bedingt (konditionale Anweisungen) oder wiederholt (Wiederholungsanweisungen bzw. Schleifen) ausgeführt.

Strukturierte Anweisung:
Blockbefehl
bedingte Anweisung
Wiederholungsanweisung

5.3.5.5 Verbundanweisungen

Die Verbundanweisungen legen fest, dass die einzelnen darin enthaltenen Komponenten in der Reihenfolge ausgeführt werden, in der sie im Quelltext erscheinen. Alle im Verbund enthaltenen Anweisungen werden als einziger Block behandelt und genügen so den Forderungen an Stellen, wo die Syntax von *rw_SymPas* nur eine einzelne Anweisung zulässt. Anfang und Ende eines Verbundes werden durch *begin* und *end* gekennzeichnet, die einzelnen Komponenten sind durch Semikolons voneinander getrennt.

Die Verbundanweisung lässt sich wie folgt darstellen:

begin Anweisungsfolge **end;**

Anweisungsfolge:
Anweisung;
Anweisungsfolge Anweisung;

Beispiel:

```
// ...
var
        i: Integer;
        j: Integer;
        temp: Integer;
// ...
begin
        if (i > 0) then i := 0;
        else begin
                // j und i vertauschen
                temp := i;
                i := j;
                j := temp;
        end;
end.
```

5.3.5.6 Bedingte Anweisungen

Bedingte Anweisungen bieten eine oder mehrere Optionen und wählen eine ihrer Komponenten (ggf. auch keine) zur Anweisung aus.

5.3.5.6.1 Die if-Anweisung

lässt sich wie folgt darstellen:

if (Bedingter-Ausdruck) **then** w-Anweisung <**else** f-Anweisung>

Die Klammern um *Bedingter-Ausdruck* sind nicht unbedingt erforderlich. Das Ergebnis von *Bedingter-Ausdruck* muss den Standardtyp *Boolean* haben. Ist *Bedingter-Ausdruck* TRUE, so wird *w-Anweisung* ausgeführt; andernfalls wird *w-Anweisung* ignoriert.
Ist das optionale *else f-Anweisung* vorhanden und *Bedingter-Ausdruck* wahr, so wird *w-Anweisung* ausgeführt; andernfalls wird *w-Anweisung* ignoriert und *f-Anweisung* ausgeführt.
Die Anweisungen *f-Anweisung* und *w-Anweisung* können selbst *if-Anweisungen* sein, wodurch verschachtelte Bedingungstest in nahezu beliebiger Tiefe ermöglicht werden. Bei verschachtelten *if..else*-Konstrukten muss sehr sorgfältig vorgegangen werden - es ist darauf zu achten, dass die korrekten Anweisungen ausgewählt werden. *Else*-Mehrdeutigkeiten werden gelöst, indem ein *else* dem letzten auf derselben Schachtelungstiefe aufgetretenen *if*-ohne-*else* zugeordnet wird. Für *w-Anweisung* und *f-Anweisung* sind auch Verbundanweisungen zulässig.

5.3.5.7 Schleifen

Schleifen (oder Wiederholungsanweisungen) legen die wiederholte Ausführung bestimmter Programmteile fest.

Schleife:
- while-Anweisung
- repeat-Anweisung
- for-Anweisung

5.3.5.7.1 Die while-Anweisung

Das Format für eine **while**-Anweisung lautet:

while (Bedingungsausdruck) **do** w-Anweisung;

Die Klammern um *Bedingungsausdruck* sind nicht unbedingt erforderlich. Die Schleifenanweisung *w-Anweisung* wird solange ausgeführt, bis der *Bedingungsausdruck* den Wert FALSE ergibt. Der *Bedingungsausdruck* wird vorher ausgewertet und getestet. Ergibt sich ein Wert, der TRUE ist (wahr), wird *w-Anweisung* ausgeführt. Wenn das Programm auf keine Sprunganweisungen trifft, die das Verlassen der Schleife zur Folge haben, wird der *Bedingungsausdruck* erneut ausgewertet. Dieser Vorgang wiederholt sich solange, bis *Bedingungsausdruck* den Wert FALSE ergibt. Gibt es keine Sprunganweisungen, muss *w-Anweisung* den Wert von *Bedingungsausdruck* beeinflussen oder *Bedingungsausdruck* selbst muss sich während der Auswertung ändern, damit Endlosschleifen vermieden werden. Für *w-Anweisung* sind auch Verbundanweisungen zulässig.

5.3.5.7.2 Die repeat-Anweisung

Das Format für eine *repeat*-Anweisung lautet:

repeat r-Anweisung **until** (Bedingungsausdruck);

Die Klammern um *Bedingungsausdruck* sind nicht unbedingt erforderlich. Die Anweisung *r-Anweisung* wird ausgeführt, solange *Bedingungsausdruck* den Wert FALSE (falsch) hat. Im Gegensatz zur *while*-Anweisung wird *Bedingungsausdruck* nicht vor, sondern nach jeder Ausführung der Schleifenanweisung getestet. *r-Anweisung* wird daher mindestens einmal ausgeführt.
Für *r-Anweisung* sind auch Verbundanweisungen zulässig.

5.3.5.7.3 Die for-Anweisung

Das Format für eine *for*-Anweisung lautet:

for Laufvariable := Startwert **to/downto** Endwert **do** f-Anweisung;

Die Laufvariable muss der Bezeichner einer Variablen des Typs *integer* sein, die entweder innerhalb desselben Blocks wie die *for*-Anweisung lokal oder global zum gesamten Programm deklariert ist. Die Definition einer Schleife mit *for* schließt die Festlegung eines *Start*- und *Endwert*es mit ein. Beide Werte müssen ebenfals vom Typ *integer* sein, der zu dem der Laufvariablen zuweisungskompatibel ist.
Beim Start der Schleife wird die Laufvariable auf den *Startwert* gesetzt und für jeden Schleifendurchlauf um eins erhöht bzw. erniedrigt - solange, bis der *Endwert* erreicht ist. Bei jedem Durchlauf wird die im Rumpf der Schleife enthaltene *f-Anweisung* bzw. Verbundanweisung einmal ausgeführt. Wenn die Endbedingung der Schleife bereits vor dem ersten Durchlauf gegeben ist (d.h. *Endwert* < *Startwert* bzw. *Endwert* > *Startwert* bei Verwendung von *downto*), dann werden Schleife und dazugehörender Rumpf komplett übersprungen.

5.3.6 Prozeduren und Funktionen

Prozeduren und Funktionen stellen formal gesehen zusätzliche Ebenen innerhalb des Hauptprogramm-Blocks dar, also eine Verschachtelung. Eine Prozedur wird durch einen Prozeduraufruf (d.h. die Angabe eines Bezeichners) aktiviert und liefert keinen direkten Wert zurück. Eine Funktion wird bei der Berechnung eines Ausdrucks aktiviert, in der ihr Bezeichner erscheint, und hat normalerweise ein Ergebnis, das für diesen Aufruf mit dem Funktionsbezeichner gleichgesetzt werden kann.

5.3.6.1 Prozedurdeklarationen

Eine Deklaration, die mit dem reservierten Wort *procedure* eingeleitet wird, verbindet einen Bezeichner und einen Block von Anweisungen zu einer Prozedur. Derartig deklarierte Prozeduren können durch Angabe ihres Bezeichners aktiviert (d.h. aufgerufen) werden. Eine Prozedurdeklaration hat folgenden formalen Aufbau:

Prozedurkopf; Prozedurblock;

Der Prozedurkopf benennt die Prozedur (d.h. ordnet ihr einen Bezeichner zu). Ab mcfg.exe V2.5.3.97 (ncc.exe/dll V2.5.3.73) ist an dieser Stelle die Deklaration von Parametern möglich. Ein Datenaustausch zwischen dem Hauptprogramm und einer Prozedur kann über globale Variablen oder über diese Parameter durchgeführt werden. Durch die Angabe ihres Bezeichners wird eine Prozedur aktiviert: Die im Befehlsteil der entsprechenden Prozedurdeklaration definierten Aktionen werden ausgeführt.
Eine Prozedur, die ihre eigene Anweisung als Bestandteil ihres Befehlsteils enthält, wird rekursiv ausgeführt, d.h. sie ruft sich selbst wiederholt auf. In diesem Zusammenhang muss ein geeignetes Kriterium für den Abbruch der Rekursion gefunden werden, bevor der interne CNC-Task-Stack überläuft.
Das Schachteln von Prozeduren in *rw_SymPas* ist nur möglich, wenn der Prozedurkopf mit optionalen Parametern als *forward* deklariert wurde. Hierzu muss nach dem Prozedurkopf das Schlüsselwort *forward*, ebenfalls gefolgt von einem Semikolon, erscheinen. Eine forward-Deklaration ist nur dann erforderlich, wenn die Prozedur vor dem Abschluss der eigentlichen Deklaration (mit Prozedurblock) verwendet werden soll. Zwischen Prozedurkopf und Prozedurblock können lokale Variablen und lokale Labels deklariert werden.
Die forward-Deklaration ist ebenfalls erst ab mcfg.exe V2.5.3.97 (ncc.exe/dll V2.5.3.73) möglich.

Beispiele:

```
procedure ProcA; forward;
...
procedure ProcA;
begin
        (Prozedurblock)
end;

procedure ProcB (ParamA, ParamB : integer; ParamC : double); forward;
...
procedure ProcB (ParamA, ParamB : integer; ParamC : double);
begin
        (Prozedurblock mit Verwendung von ParamA, ParamB und ParamC)
end;

procedure ProcC (ParamA : integer);
var locVarA, locVarB : integer;
    locVarC : double;
begin
        (Prozedurblock)
end;
```

5.3.6.2 Funktionsdeklarationen

Anmerkung: Die nach *Pascal*-Standard implementierte Funktionendeklaration ist in *rw_SymPas ab* V2.5.3.97 (ncc.exe/dll V2.5.3.73) möglich. Zur Ausführung ist rwmos.elf ab V2.5.3.126 erforderlich, ansonsten wird ein rw_SymPas Programm mit dem Laufzeitfehler 4 beim Rücksprung aus der Funktion beendet. Außerdem gibt es verschiedene vordefinierte System-Funktionen.

Die Funktionen (Systemfunktionen und anwenderdefinierte Funktionen) werden bei der Berechnung von Ausdrücken aktiviert, in denen ihr Bezeichner erscheint, und stehen dort stellvertretend für den von ihnen zurückgelieferten Wert. Ein Funktionsbezeichner kann überall anstelle eines Operanden in einem Ausdruck eingesetzt werden, solange der Typ des Funktionsergebnisses mit dem des ersetzten Operanden kompatibel ist. Zuweisungen an einen Funktionsbezeichner sind nicht erlaubt. Der Aufruf einer Funktion geschieht über die Angabe ihres Bezeichners, gefolgt von einer Liste aktueller Parameter, die in Typ und Reihenfolge den formalen Parametern der enstprechend vordefinierten Funktion entsprechen müssen. Der Rückgabewert einer Funktion muss verwendet werden; ansonsten wird der Übersetzungsfehler 91 angezeigt.

Die Deklaration einer anwenderspezifischen Funktion, die mit dem reservierten Wort *function* eingeleitet wird, verbindet einen Bezeichner, Typdeklarationen und einen Block von Anweisungen zu einer Funktion. Derartig deklarierte Funktionen können durch Angabe ihres Bezeichners aktiviert (d.h. aufgerufen) werden. Eine Funktionsdeklaration hat folgenden formalen Aufbau:

Funktionskopf; Funktionsblock;

Der Funktionskopf benennt die Funktion (d.h. ordnet ihr einen Bezeichner zu) und definiert Parameter und Funktionstyp. Ein Datenaustausch zwischen dem Hauptprogramm und einer Funktion kann über globale Variablen oder über Parameter und den Rückgabewert durchgeführt werden. Durch die Angabe ihres Bezeichners wird eine Funktion aktiviert: Die im Befehlsteil der entsprechenden Funktionsdeklaration definierten Aktionen werden ausgeführt.
Eine Funktion, die ihre eigene Anweisung als Bestandteil ihres Befehlsteils enthält, wird rekursiv ausgeführt, d.h., sie ruft sich selber wiederholt auf. In diesem Zusammenhang muss ein geeignetes Kriterium für den Abbruch der Rekursion gefunden werden, bevor der interne CNC-Task-Stack überläuft.

Das Schachteln von Funktionen in *rw_SymPas* ist nur möglich, wenn der Funktionskopf mit optionalen Parametern und Funktionstyp als *forward* deklariert wurde. Hierzu muss nach dem Funktionskopf das Schlüsselwort *forward*, ebenfalls gefolgt von einem Semikolon, erscheinen. Eine forward-Deklaration ist nur dann erforderlich, wenn die Funktion vor dem Abschluss der eigentlichen Deklaration (mit Funktionsblock) verwendet werden soll. Im Funktionsblock muss der Rückgabewert einer Variablen mit der Bezeichnung des Funktionsnamens zugewiesen werden. Diese Variable für den Rückgabewert kann innerhalb der Funktion als lokale Variable verwendet werden. Wenn ein rekursiver Aufruf der Funktion erfolgen soll, muss der Funktionsname mit einem runden Klammernpaar, welches die Parameter enthält, verwendet werden. Bei Funktionen ohne Parameter muss in diesem Fall der Funktionsaufruf durch ein leeres rundes Klammernpaar gekennzeichnet werden.
Zwischen Funktionskopf und Funktionsblock können lokale Variablen und lokale Labels deklariert werden (siehe Beispiel bei den Prozeduren).

Beispiele:

```
function FuncA  : integer; forward;
...
function FuncA  : integer;
begin
        (Funktionsblock)
        FuncA := Funktionsrückgabewert;
end;

function FuncB (ParamA, ParamB : integer; ParamC : double) : double; forward;
...
function FuncB (ParamA, ParamB : integer; ParamC : double) : double; forward;
begin
        (Funktionsblock mit Verwendung von ParamA, ParamB und ParamC)
        FuncB := Funktionsrückgabewert;
end;
```

Funktionsrückgabewert kann in obigen Beispielen ein beliebiger Ausdruck sein.

5.3.7 Die Syntax eines *rw_SymPas*-Programms

Ein *rw_SymPas*-Programm hat eine ähnliche Form wie eine Prozedurdeklaration. Die Unterschiede liegen lediglich im Programmkopf.

rw_SymPas-Programm:

Programmkopf; Programmblock**.**

5.3.7.1 Der Programmkopf

Der Programmkopf legt den Namen eines Programms fest, hat aber keine besondere Bedeutung.

Programmkopf:
program Bezeichner

Beispiel:

```
program Test;
```

5.3.7.2 Der Programmblock

Programmblock:
Implementationsteil
Prozedur-/Funktions-Befehlsteil
Initialisierungsteil

Implementationsteil:
Konstanten-Deklaration
Variablen-Deklaration
Implementationsteil Konstanten-Deklaration
Implementationsteil Variablen-Deklaration

Initialisierungsteil:
begin
Befehlsteil
end

Der Initialisierungsteil ist der letzte Bestandteil eines *rw_SymPas*-Programms und stellt das Hauptprogramm dar. Er besteht aus einem mit **begin** eingeleiteten Block, der Anweisungen enthält und durch ein abschließendes **end** beendet wird. Der gesamte Programmblock wird mit dem Zeichen **„.**“ abgeschlossen.

6 Standalone-Applikations-Programmierung

6.1 Einführung

Die Programmiersprache *rw_SymPas* ist mit einem umfangreichen Befehlssatz ausgestattet, mit dessen Hilfe eine flexible und effiziente Programmerstellung ermöglicht wird. Die Prozeduraufrufe werden bis auf wenige Ausnahmen nach *Pascal*-Konvention durchgeführt.
Da die Prozedurnamen und auch die Funktionsweise der einzelnen Prozeduren für die beiden Programmiermethoden Standalone-Applikations-Programmierung [SAP] und PC-Applikations-Programmierung [PCAP] identisch sind, erfolgt eine detaillierte Beschreibung nur bei den Befehlen der PCAP-Programmierung.
Die Auflistung der einzelnen Befehle erfolgt in alphabetischer Reihenfolge.

6.2 rw_SymPas-Beispielprogramme

Die in der APCI-800x TOOLSET Software enthaltenen *rw_SymPas* Beispielprogramme zeigen die einfache Anwendung nachfolgend beschriebener Funktionen. Die Quelltexte der Beispielprogramme sind durch Kommentare selbsterklärend. Deshalb wird an dieser Stelle auf eine detaillierte Programmbeschreibung dieser Beispiele verzichtet. Die Beispielprogramme haben alle den Dateierweiterungsnamen .SRC und sind im Unterverzeichnis SAP der APCI-800x TOOLSET Software Diskette abgelegt.

6.3 Abkürzungen, System-Parameter, Achsenspezifizierer und Achsenqualifizierer

Für die nachfolgend abgedruckte SAP-Funktionenreferenzliste werden hier zunächst die einzelnen Abkürzungen und Typen erklärt, welche zum Teil als Parameter für die verschiedenen Funktionen dienen.

6.3.1 System-Parameter

Die von der *rw_SymPas*-Programmiersprache vordefinierten Systemparameter werden tabellarisch aufgelistet und deren Funktionsweise erläutert. Zu beachten ist, dass der *NCC*-Compiler bei diesen Parametern zwischen Groß- und Kleinbuchstaben unterscheidet.

Tabelle 32: *rw_SymPas* vordefinierte System-Parameter

Name	Typ	Kurzwortbedeutung	Funktion
BOARD TYPE	integer	Board-Typ	Hardwarevariante der Steuerungstyps (siehe Kapitel 4.4.18)
CI0..CI999	integer	Common Integer 0..999	1000 vordefinierte Integer-Variablen zum Datenaustausch oder zur Synchronisation mit einem parallel ablaufendem PC-ApplikationsProgramm. Weitere Infos bei den PCAP-Befehlen rdci() u. wrci().
CD0.. CD999	double	Common Double 0..999	1000 vordefinierte Double-Variablen. Sonst wie CI0..CI999. Weitere Informationen bei den PCAP-Befehlen rdcd() und wrcd().
CFLAG	integer	ControllerFlags	Zugriff auf das Register ControllerFlags (siehe Kapitel 6.3.1.4)
DTCA1	double	Distance-to-Center A1	Mittelpunktsangabe für Helix-Profile und 3D-Kreise für die X-Kreisachse
DTCA2	double	Distance-to-Center A2	Mittelpunktsangabe für Helix-Profile und 3D-Kreise für die Y-Kreisachse
DTCA3	double	Distance-to-Center A3	Mittelpunktsangabe für 3D-Kreise für die Z-Kreisachse
ERROR REG	integer	error register	Bitcodiertes Fehlerregister, in welchem interne Fehlerzustände von RWMOS.ELF angezeigt werden.
GFAUX	double	Gear Faktor Aux-Register	Angabe eines Übersetzungsverhältnis zwischen aux-Register und Sollposition für Systeme mit Puls-Richtungs-Schnittstelle
IRQPC	boolean	Interrupt Request PC	PC-Interrupt-Anforderung, aktiv wenn TRUE
LEDGN	boolean	Led green	Grüne Leuchtdiode auf APCI-800x, eingeschaltet wenn TRUE
LEDRD	boolean	Led red	Rote LED, sonst wie LEDGN
LEDYL	boolean	Led yellow	Gelbe LED, sonst wie LEDGN
LET	double	Latch End Time	Zeitpunkt für die Dauer der Aufzeichnung für die grafische Systemanalyse in Sekunden, vom Zeitpunkt LST an. Siehe hierzu die Befehle LPR und LPRS. Grundsätzlich werden immer 1000 Werte aufgezeichnet. Der in LET angegebene Wert wird immer in ganzzahlige Vielfache von 1000 * T_A aufgerundet. T_A ist standardmässig 1.28ms.
LST	double	Latch Start Time	Zeitpunkt für den Beginn der Aufzeichnung für die grafische Systemanalyse in Sekunden, vom Zeitpunkt des Aufrufs an. Siehe hierzu die Befehle LPR und LPRS.
MODEREG	integer	Mode Register	Bitcodiertes Register zur Steuerung der Betriebssystem-Funktionalität (siehe Kapitel 6.3.1.5)
NFRAX	integer	No-Feed-Rate-Axis	In dieser Variable können Achsen bitcodiert definiert werden, die bei Interpolationsbewegungen nicht für die Bahngeschwindigkeitsberechung herangezogen werden.

Name	Typ	Kurzwortbedeutung	Funktion
NOA	integer	Number of Axis	Diese Systemvariable enthält die Anzahl der tatsächlich im System vorhandenen Achsen und darf nicht beschrieben werden.
OS VERSION	integer	Betriebssystem–Versionsinformation	Mit dem vordefinierten Systemparameter *OSVERSION* (Typ) kann die aktuelle Betriebssystemversionsnummer des *rwmo.elf*-Files zur Laufzeit in einem SAP-Programm abgefragt werden. Die Versionsnummer ist in eine Haupt und eine Nebennummer gegliedert, wobei die Hauptnummer in 1000er Schritten und die Nebennummer in 1er Schritten hoch gezählt wird. Die Versionsnummer 253042 z.B. bedeutet dann Hauptnummer 2.5.3 und Nebennummer 042.
PHI	double		Verfahrwinkel für Kreis- und Helix-Profile
PN1	double	Plane-Normal	Flächen-Normale für MCA3D Kommando. Weitere Informationen beim Kommando MCA3D.
PN2	double	Plane-Normal	Flächen-Normale für MCA3D Kommando. Weitere Informationen beim Kommando MCA3D.
PN3	double	Plane-Normal	Flächen-Normale für MCA3D Kommando. Weitere Informationen beim Kommando MCA3D.
PU	integer	Position Unit	Index für Positions-Einheit (Tabelle 33)
SSFP	integer	Spool-Special-Function-Parameter	Funktionsparameter für Spezialfunktionen in der Spooler-Betriebsart. Weitere Informationen beim SAP-Kommando *SSF*
TRAC	double	Trajectory Acceleration	Bahnbeschleunigung für Linear-, Kreis- und Helix-Profile. Die Einheit dieses Parameters ist in TU und PU festgelegt.
TROVR	double	Trajectory Override	Bahngeschwindigkeitskorrekturwert
TROVRST	double	Trajectory Override Settling Time	Zeit für Anpassung des Bahngeschwindigkeitskorrekturwertes
TRTVL	double	Trajectory Target Velocity	Bahnzielgeschwindigkeit für Linear, Kreis- und Helix-Profile. Die Einheit dieses Parameters ist in TU und PU festgelegt.
TRVL	double	Trajectory Velocity	Bahngeschwindigkeit für Linear, Kreis- und Helix-Profile. Die Einheit dieses Parameters ist in TU und PU festgelegt.
TU	integer	Time Unit	Index für Zeit-Einheit (Tabelle 34)

6.3.1.1 PC-Interrupt-Generierung

Mit Hilfe des System-Parameters *IRQPC* ist es möglich, auf dem PC einen Hardware-Interrupt auszulösen. Diese Möglichkeit gestattet eine effiziente Methode für den Einsatz der beiden Programmiermethoden PC-Applikations- und Standalone-Applikations-Programmierung. Mit Hilfe eines Stand-Alone-Programms kann ein weitgehend autarker Prozessablauf erfolgen, der nur im Bedarfs- oder Fehlerfall das parallel ablaufende PC-Programm unterbrechen muss. Die Unterbrechung geschieht dann mit Hilfe dieser Interrupt-Generierung. Nach Erkennen des Hardware-Interrupts durch das PC-Programm könnte dann z.B. mit Hilfe der oben aufgeführten Common Variablen ein Datenaustausch zwischen den beiden parallel ablaufenden Programmen erfolgen.

Anmerkung zur PCI-Interruptverwaltung: Der Hardware-Interrupt (PCI-Interrupt) wird dank der auf der APCI-800x integrierten Plug and Play-Eigenschaften automatisch ermittelt und durch den Systemtreiber mcug3.dll verwaltet. Der Anwender muss lediglich eine PCAP-Benutzerroutine mit vorgegebenem Aufbau definieren und diese dem Treiber bekannt machen. Sobald die APCI-800x einen Hardware-Interrupt auslöst, wird die entsprechende Benutzerroutine automatisch aufgerufen. Der mcug3.dll-Treiber ist so gestaltet, dass auch andere PCI-Interrupts, die die gleiche Interruptquelle benutzen, aufgerufen werden. Die im Lieferumfang enthaltene Treibersoftware stellt Funktionen bereit um auf einfachem Wege eine Interrupt-Service-Routine zu installieren und bei Bedarf auch wieder zu deinstallieren (Kapitel 4.4.33 und 4.4.34).

6.3.1.2 Systemparameter für die Einheiten-Verarbeitung

Bei sämtlichen *move*-Befehlen der *rw_SymPas*-Programmiersprache erfolgt die Angabe der Beschleunigungs- *(TRAC)*, Geschwindigkeits- *(TRTVL, TRVL)* und Positionsparameter jeweils in angewählten Weg- und Zeiteinheiten. Mit den beiden nachfolgend aufgeführten Systemparametern ist es möglich, die Weg- (PU) und Zeit-Einheit (TU) jederzeit umzuschalten.

Tabelle 33: System-Parameter PU

Wert	Einheit	Kurzwortbedeutung
0	mm	Millimeter
1	inch	Inch
2	m	Meter
3	rev	Revolution
4	deg	Degree
5	rad	Radiant
6	counts	Counts
7	steps	Steps

Tabelle 34: System-Parameter TU

Wert	Einheit	Kurzwortbedeutung
0	sec	Seconds
1	min	Minutes
2	tsample	Sampling Time

Anmerkung: Die Standardwerte für TU und PU werden in dem Menü [Setup][Set CNC-specific parameters] in der CNC-Editor-Umgebung festgelegt.
Die gewählten Einheiten werden nur für Interpolationsbefehle (alle *move*-Befehle) verwendet! Sofern es sich um achsspezifische Bewegungskommandos (alle *jog*-Befehle) handelt, werden die in *mcfg.exe* spezifizierten Achs-Einheiten berücksichtigt. Hier ist keine Umschaltung während der Laufzeit möglich.

6.3.1.3 ERRORREG

In diesem bitcodierten Register werden Laufzeitfehler der RWMOS-Betriebssystemsoftware angezeigt.
Die Belegung der Bits kann Tabelle 15 in Kapitel 4.4.63.1 entnommen werden.

Das Register ERRORREG wird nur durch Beschreiben mit 0 oder durch einen Systemboot zurückgesetzt.

6.3.1.4 ControllerFlags

Dies ist ein achsspezifisches bitcodiertes Register, mit dessen Hilfe sich Spezialoptionen im Lageregler der Achsensteuerungskarten aktivieren lassen. Es gibt Optionen für das Verhalten des Lagereglers während des Verfahrens und im Stillstand. Dieses Register ist nur bei Servo-Achsen mit PID-Filtercharakteristik wirksam; bei Stepper-Achsen hat es keine Wirkung. Auf das Register kann mit den dll-Funktionen wrControllerFlags (Kapitel 4.4.137) und rdControllerFlags (Kapitel 4.4.51) zugegriffen werden. Aus der rw_SymPas Programmierung kann auf das Register mit dem Achsenqualifizierer CFLAGS zugegriffen werden.

Tabelle 35: Beschreibung des Registers ControllerFlags

Bit # / Hex	Bez.	Erläuterung
0 / 0001H	***MoveControl***	Aktivierung aller Flags Move...
1 / 0002H	***MoveZero***	Ablöschen des Integralanteils des Achsreglers, sobald die Achse verfahren wird
2 / 0004H	***MoveDeadBand***	Integralanteil des Achsreglers begrenzen, sobald die Achse verfahren wird und sich die Regeldifferenz der Achse außerhalb des Totbandes befindet. Das Totband wird im Feld ControllerParams [8][0] in Digits angegeben (siehe auch Kapitel 4.4.114)
3 / 0008H	***MoveFix***	Integralanteil des Achsreglers festhalten und nicht weiter aufintegrieren, sobald die Achse verfahren wird
4 / 0010H		not used
5 / 0020H		not used
6 / 0040H		not used
7 / 0080H		not used
8 / 0100H	***StopControl***	Aktivierung aller Flags Stop...
9 / 0200H	***StopZero***	Ablöschen des Integralanteils des Achsreglers, sobald die Achse steht
10 / 0400H	***StopDeadBand***	Integralanteil des Achsreglers begrenzen, sobald die Achse steht und sich die Regeldifferenz der Achse außerhalb des Totbandes befindet. Das Totband wird im Feld ControllerParams [8][0] in Digits angegeben. (siehe auch Kapitel 4.4.114)
11 / 0800H	***StopFix***	Integralanteil des Achsreglers festhalten und nicht weiter aufintegrieren, sobald die Achse steht
12 - 31		not used

Move... bedeutet, dass die entsprechenden Flags ihre Wirkung entfalten, sobald die entsprechende Achse verfährt. Stop... bedeutet, dass die entsprechenden Flags ihre Wirkung entfalten, sobald die entsprechende Achse sich in Lageregelung befindet. Wenn StopControl und MoveControl auf FALSE gesetzt sind, sind alle anderen Bits unwirksam. Zur Anzeige und zur testweisen Einstellung dieser Flags gibt es das Hilfsprogramm McuControlInit.exe, welches im Bedienungshandbuch (BHB) beschrieben wird.

6.3.1.5 MODEREG

Mit Hilfe dieses bitcodierten Registers können diverse Optionen der Betriebssystemsoftware RWMOS.ELF eingestellt werden. Standardmäßig sind alle Bits auf 0 eingestellt.

Hinweis: Wenn in diesem Register ein Bit gesetzt oder rückgesetzt werden soll, muss im Allgemeinen darauf geachtet werden, dass die anderen Bits nicht verändert werden. Dazu ist es notwendig, vor Schreiben auf MODEREG, dieses zunächst zu lesen, den Inhalt mit Hilfe boolescher Operationen zu bearbeiten und dann wieder zurückzuschreiben. Setzen von Bits führt man mit boolescher Oder-Verknüpfung, Rücksetzen von Bits mit boolescher Und-Verknüpfung durch. Bits, die momentan nicht belegt sind, dürfen nicht verwendet werden, da diese für zukünftige Erweiterungen reserviert sind.

Tabelle 36: Bitcodierung MODEREG

Bit # / Hex	Bez.	Erläuterung
0 / 0001H	***LookAhead***	Mit diesem Bit wird die Look-Ahead-Funktionalität der RWMOS-Betriebssystemsoftware aktiviert. Hierbei wird die angegebene Zielgeschwindigkeit von Interpolationsprofilen so begrenzt, dass der maximale achsspezifische Geschwindigkeitssprung MDVEL bei keiner Achse überschritten wird und dass am Ende der Interpolationsfahrt alle Achsen stehen. Dieser Modus bezieht sich nur auf die Kommandos SMLA, SMLR, SMCA, SMCR, SMHA, SMHR. Des Weiteren wird in diesem Modus die Bahngeschwindigkeit in Kreis-Befehlen so begrenzt, dass bei keiner der beteiligten Achsen die achsspezifische Maximum-Acceleration (MaxAcc) überschritten wird. Die maximale Geschwindigkeit ergibt sich aus √ (Kreisradius * >MaxAcc). Siehe hierzu auch Kapitel 4.4.160, 4.4.85 und 6.3.3.
1 / 0002H	***S-Profil***	Durch Setzen dieses Bits werden die Beschleunigungs- und Bremsrampen mit S-förmigem Geschwindigkeitsanstieg / -abfall ausgeführt. Diese Option kann mit dem Achsenqualifizierer JERKREL parametriert werden.
2 / 0004H		frei für zukünftige Verwendung
3 / 0008H	***WkzRadKorr***	Werkzeug-Radius-Korrektur ein (nur bei Option TC) Zur Werkzeug-Radius-Korrektur existiert ein gesondertes Handbuch.
4 / 0010H		frei für zukünftige Verwendung
5 / 0020H	***AutoSpool***	Wenn Verfahrprofile in SAP-Programmen gespoolt werden, wird bei aktiver Option jeweils der Spooler geprüft. Wenn der Spooler voll ist, wird bei den im aktuellen Profil selektierten Achsen die Spoolerabarbeitung automatisch gestartet. Weitere Profile werden nur eingetragen, wenn wieder Speicherplatz frei ist. Diese Option wird für folgende Verfahrbefehle unterstützt: SMLA, SMLR, SMCR, SMCA, SMHA, SMHR und G01 bei DIN66025
6 / 0040H	***NoTriangle***	Dreiecksprofile im LookAhead-Modus werden unterdrückt. Falls ein Teilprofil die programmierte Bahngeschwindigkeit nicht erreichen kann, wird die aktuelle Startgeschwindigkeit beibehalten. Dadurch wird bei kurzen Verfahrprofilstücken ein glatterer Lauf ohne ständige Beschleunigungs- und Bremsphasen erreicht.
7 / 0080H	***ChkMaxVel***	In diesem Modus wird bei gespoolten Linear-Interpolationsbefehlen die Bahngeschwindigkeit und die Bahnbeschleunigung aller Achsen so begrenzt, dass keine Achse die in MAXVEL und MAXACC gesetzen Maximalwerte überschreitet. In diese Überwachung werden auch No-Feedrate-Achsen mit einbezogen (siehe Tabelle 32 – NFRAX).
8 / 0100H	***ExactTargetPos***	Normalerweise wird am Ende eines jeden Verfahrprofils, welches die Zielgeschwindigkeit 0 hat, die Sollposition auf ganzzahlige Werte der Systemauflösung gerundet. Dies ist bei Schrittmotorsystemen z.B. ein Schritt oder bei Enkodersystemen ein Enkoder-Zählimpuls. Dadurch kann sich, beim hintereinanderhängen von Relativprofilen ein Fehler aufsummieren. Diese Rundung kann durch Setzen dieses Bits abgeschaltet werden.
9 / 0200H	***ShortestRotatoric Distance***	Wenn dieses Bit gesetzt ist, werden bei rotatorischen Achsen Jog-Absolut-Befehle (JA) in die Richtung ausgeführt, in die der kürzeste Verfahrweg notwendig ist.
10 / 0400H	***RotatoricUnit***	Wenn dieses Bit gesetzt ist, wird bei rotatorischen Achsen, die mit translatorischen Achsen per Interpolationsbefehl verfahren werden sollen, die Zielposition / der Verfahrweg in der achsspezifischen rotatorischen Einheit angegeben.

Bit # / Hex	Bez.	Erläuterung
11 / 0800H	*ForbidTargetVel*	Wenn dieses Bit gesetzt ist, wird ein System-Reset (rs) durchgeführt, wenn Verfahrprofile mit einer Zielgeschwindigkeit <> 0 beendet werden. In diesem Fall wird in der Systemvariablen ErrorReg das Bit 1 gesetzt.
12 / 1000H	*CenterAlwaysRel*	Kreismittelpunkte bei G-Codes G02 und G03 immer als Relativkoordinaten interpretieren
13 / 2000H	*NoLsmCheck*	Durch Setzen dieses Flags kann die automatische Spoolerüberwachung bei G01 des G-Code-Interpreters (McuWIN) abgeschaltet werden.
14 / 4000H		frei für zukünftige Verwendung
15 / 8000H	*MS_DECEL*	Beim Kommando MotionStop (ms) als Bremsbeschleunigung den Wert von TRAC verwenden unter Berücksichtigung der aktiven Interpolationseinheiten
16 / 1 0000H bis 23/80 0000H		frei für zukünftige Verwendung
24 / 0100 0000H	*SimulationMode*	Mit diesem Bit kann die Steuerung in den Simulationsmodus versetzt werden. In diesem Modus, werden keine Stellgrößen an die Antriebssysteme ausgegeben, der Verlauf der Istposition wird simuliert. **Vorsicht:** Eine Drift der Achsen muss vom Anwender verhindert werden, da in diesem Modus der Lageregler nicht aktiv ist.
25 / 0200 0000H	*OvrMode*	Wenn dieses Bit gesetzt ist, wird beim Aufruf des Kommandos ctru der Jog-Override der selektierten Achsen nicht beeinflusst.
26 / 0400 0000H	*StopAtWriteln*	Wenn dieses Bit gesetzt ist, bewirkt das SAP-Kommando writeln einen Stopp der jeweiligen Task. In diesem Fall wird im Register running der Datenstruktur CNCTS (Kapitel 4.3.2.10) zusätzlich das Bit 2 gesetzt. Dieser Modus kann zur lückenlosen Verarbeitung von Ausgabestrings in einem überlagerten Programm verwendet werden.
27 / 0800 0000H	*ClearZeroPosition*	Wenn dieses Bit gesetzt ist, wird die mit szpa / szpr gesetzte Nullpunktverschiebung gelöscht. Die aktuellen Positionswerte bleiben jedoch erhalten. Bei gesetztem Bit kann somit z.B. jederzeit die Referenzposition mit szpr verschoben werden.
28 / 1000 0000H	*JSatSAF*	JOG-Stop at Spooler-Asynchronous-Flag: Wenn dieses Bit gesetzt ist, werden bei Auftreten eines SAF-Flags im AXST-Register alle Achsen mit der programmierten Stop Deceleration angehalten. Im Fehlerfall wird auch Bit 19 im ErrorReg gesetzt.
29 / 2000 0000H	*InhibitProfile Refuse*	Normalerweise werden Interpolations-Verfahrprofile ohne Verfahrweg oder mit Geschwindigkeit/Beschleunigung = 0 automatisch verworfen und eine Fehlermeldung im fwsetup Monitor-Screen wird generiert. Mit diesem Bit kann die Ausgabe einer Fehlermeldung beim Verwerfen von Verfahrprofilen unterdrückt werden.
folgende Bits		derzeit nicht belegt, reserviert für zukünftige Verwendung

6.3.2 Achsen-Spezifizierer

Die verschiedenen Achskanäle werden mit einem symbolischen Namen referenziert. Diese Namen können im Programm *mcfg.exe* vom Anwender frei gewählt werden. In der Programmiersprache *rw_SymPas* werden diese Namen automatisch vordefiniert und dienen im Anwenderprogramm bei verschiedenen Befehlen als Parameter. Zu beachten ist, dass der *NCC*-Compiler bei den Achsen-Spezifizierern zwischen Groß- und Kleinschreibung unterscheidet.

6.3.3 Achsen-Qualifizierer

Die nachfolgend aufgeführten Systemparameter verstehen sich als Achsen-Qualifizierer und sind deshalb für alle im System vorhandenen Achskanäle und damit für alle Achsen-Spezifizierer verfügbar. Mit Hilfe dieser Parameter können verschiedene achsspezifische Daten abgefragt bzw. gesetzt werden. Zu beachten ist, dass der *NCC*-Compiler bei diesen Parametern zwischen Klein- und Großschreibung unterscheidet. Der Bezug auf einen Achsen-Qualifizierer wird durch Angabe eines Achsen-Spezifizieres, das Zeichen '.' und dem Achsen-Qualifizierer hergestellt. Dies wird mit nachfolgendem Beispiel ersichtlich:

```
...
var
        input: integer;
...
input := A2.digi;           // Digitaleingänge von Achskanal 2
                                                    // einlesen
...
```

Tabelle 37: Achsen-Qualifizierer

Name	Typ	Kurzwortbedeutung	Funktion
an	integer	axis number	Der Achsenqualifizierer an beinhaltet die Achsnummer des entsprechenden Achsenbezeichners. Der Qualifizierer kann im Zusammenhang mit „variablen" Achsnamen verwendet werden. [Kapitel 5.3.3.1.1]
aux	double	Auxiliary Register	Der Inhalt dieses Registers ist optionsabhängig. Falls das System die optionEV (Encoder-Verification) enthält, kann über diese Variable auf den Enkoder-Zählerstand bei Schrittmotorsystemen zugegriffen werden. In diesem Fall ist die Einheit des Registers Counts.
axst	integer	axis status	Error-, Status- und Profil-Flags (wortweise)
digi	integer	digital inputs	Digital-Eingänge der APCI-800x (wortweise) Verschiedene Flags dieses Registers können durch Zuweisung eines beliebigen Werts an dieses Register gelöscht werden [Kapitel 4.4.51.1]
digo	integer	digital outputs	Digital-Ausgänge der APCI-800x (wortweise)
dp	double	desired position	Soll-Position des Achskanals
dpoffset	double	desired position offset	In diesem Register kann ein Positions-Offset für den Lageregler in der achsspezifischen Benutzereinheit eingetragen werden. Dieses Register kann für eine überlagerte Lageregelung, z.B. bei Steppern mit Enkoderverifikation, verwendet werden. Dieses Register steht für Steppersysteme ab RWMOS-Version 2.5.2.23, für Servosysteme ab RWMOS-Version 2.5.2.29 zur Verfügung.
dv	double	desired velocity	Soll-Geschwindigkeit des Achskanals
dvoffset	double	desired velocity offset	In diesem Register kann eine Änderungs-geschwindigkeit des Positions-Offset (dpoffset) für den Lageregler in der achsspezifischen Benutzereinheit eingetragen werden.
effradius	double	Effektiv Radius	Bei Teilnahme von rotatorischen Achsen an translatorischen Interpolationsfahrten: achsenspezifischer Parameter für Umrechung von rotatorischen in translatorische Größen, (Mantelflächenbearbeitung) [Kapitel 2.3.4]

Name	Typ	Kurzwortbedeutung	Funktion
epc	integer	EEPROM programming cycles	Anzahl der Programmierzyklen
gcr	integer	gear configuration register	Mit Hilfe dieses Registers, kann die Gear-Funktionalität der APCI-8001 gesteuert werden. Die Verwendung dieses Registers wird auch bei der Beschreibung der Gear-Funktionalität im Dokument „Ressourcen-Interface“ beschrieben.
gf	double	gear factor	Über diese Variable kann auf den achsspezifischen Getriebefaktor zugegriffen werden. Eine Zuweisung an diesen Wert darf nur in besonderen Fällen erfolgen.
gfaux	double	Gear Faktor Aux-Register	Angabe eines Übersetzungsverhältnisses zwischen aux-Register und Sollposition für Systeme mit Puls-Richtungs-Schnittstelle **Beispiel:** 1000 Schritte / Umdrehung Encoder mit 500 Strichen ergibt mit Vervierfachung 2000 Impulse pro Umdrehung. Hier muss der Wert 0,5 in gfaux eingetragen werden.
ifs	integer	interface status	Interface Status-Flags der APCI-800x (wortweise) Verschiedene Flags dieses Registers können durch Zuweisung eines beliebigen Werts an dieses Register gelöscht werden [Kapitel 4.4.69.1]
hac	double	home acceleration	Beschleunigung für home-Befehle
hvl	double	home velocity	Geschwindigkeit für home-Befehle
ipw	double	in position window	Positionsabhängiges Zielfenster
jac	double	jog acceleration	Beschleunigung für jog-Befehle
jerkrel	double	Jerk Relativ	Parameter für S-Geschwindigkeitsprofile
jovr	double	jog override	Geschwindigkeitsfaktor
jtvl	double	jog target velocity	Zielgeschwindigkeit für jog-Befehle
jvl	double	jog velocity	Geschwindigkeit für jog-Befehle
kd	double		PIDF-Filter-Koeffizient für Differentation
kfca	double		PIDF-Filter-Koeffizient zur Vorwärtskompensation für Beschleunigung
kfcv	double		PIDF-Filter-Koeffizient zur Vorwärtskompensation für Geschwindigkeit
ki	double		PIDF-Filter-Koeffizient für Integration
kp	double		PIDF-Filter-Koeffizient für Verstärkung
kpl	double		PIDF-Filter-Koeffizient zur zus. Phasen-Voreilung
lp	double	latched position	gelatchter Positionswert
lpndx	double	latched position index	gelatchter Positionswert mit Index-Signal (Nullspur)
lsm	integer	left spool memory	freier Spoolbereich [Bytes]
maxacc	double	maximum acceleration	Achsspezifische Maximalbeschleunigung im ChkMaxVel-Modus
maxvel	double	maximum velocity	Achsspezifische Maximalgeschwindigkeit im ChkMaxVel-Modus
mcis	integer	Move Commands in Spooler	In diesem Register wird angezeigt, wie viele Verfahrkommandos derzeit im Spooler enthalten sind. Dadurch kann der Abarbeitungszustand des Spoolers festgestellt werden. Diese Information kann verwendet werden, wenn der aktuelle Bearbeitunszustand nach Unterbrechung fortgesetzt werden soll (siehe auch PCAP-Kommando rdMCiS).

Name	Typ	Kurzwortbedeutung	Funktion
mcp	integer	Motor Command Port	Servo-Motoren: Sollwert für Analog-Port Stepper-Motoren: Schrittsignal für Schrittmotor-Leistungsendstufen Zusätzliche Beschreibung bei den Kommandos wrmcp (Kapitel 4.4.152) und rdmcp (Kapitel 4.4.87).
mdvel	double	maximum velocity skip	Achsspezifischer maximaler Geschwindigkeitssprung im Look-Ahead-Modus
mpe	double	maximum position error	maximal erlaubter Schleppfehler
poserr	double	position error	In diesem Register wird der aktuelle achsspezifische Schleppfehler in der Benutzereinheit angezeigt. Dies ist der in Echtzeit berechnete Wert dp – rp.
pprev	double	Pulses Per Revolution	In diesem Register kann die Anzahl der Encoder-Impulse pro Umdrehung (antriebsseitig) gelesen werden. Bei Stepper- und Linearachsen bzw. wenn die Nennereinheit von slsp eine Lineareinheit ist, wird hier 0 zurückgeliefert. Gegenüber slsp ist hier eine eventuelle Impulsvervierfachung berücksichtigt.
rp	double	real position	Ist-Position des Achskanals
rv	double	real velocity	Ist-Geschwindigkeit des Achskanals [Kapitel 4.4.97], kann nur gelesen, nicht zugewiesen werden
sdec	double	stop deceleration	Stopverzögerung des Achskanals
sf	integer	special function	Applikationsspezifisches Sonderregister.
sll	double	software limit left	linke Software-Endlage
slr	double	software limit right	rechte Software-Endlage
slsp	double	Slits or Stepper Pulses	In diesem Register kann die Anzahl der Enkoderstriche pro Umdrehung (antriebsseitig) bzw. die Anzahl der Schritte pro Umdrehung bei Schrittmotoren gelesen und gesetzt werden. Vervierfachung und Einheiten entsprechen den in mcfg gesetzten Werten.
tp	double	target position	Ziel-Position des Achskanals
zerooffset	double	Zero-Offset	Absolutwert der Nullpunktverschiebung siehe auch PCAP-Kommandos szpa und szpr bzw. azo

Eine detaillierte Funktionsbeschreibung dieser Qualifizierer (Tabelle 37) kann bei den entsprechenden *rdxxxx()*- bzw *wrxxxx()*-Befehlen in der Funktionenreferenzliste der PCAP-Programmierung nachgelesen werden. *Beispielsweise wird die Bedeutung des Qualifizierers digo beim Befehl wrdigo() erklärt.*

Ausnahme: Die PIDF-Filterkoeffizienten werden gemeinsam mit dem SAP-Befehl *UF()* wirksam. Das Lesen bzw. Beschreiben dieser Koeffizienten erfolgt auf PCAP-Ebene mit den Befehlen *rdf()* und *uf()*.

6.3.4 Strukturierte Achsen-Qualifizierer

Die nachfolgend aufgeführten Systemparameter verstehen sich als strukturierte Achsen-Qualifizierer und sind deshalb für alle im System vorhandenen Achskanäle und damit für alle Achsen-Spezifizierer verfügbar. Mit Hilfe dieser Parameter können verschiedene achsspezifische Daten bitweise abgefragt bzw. gesetzt werden. Zu beachten ist, dass der *NCC*-Compiler bei diesen Parametern zwischen Klein- und Großschreibung unterscheidet. Der Bezug auf einen strukturierten Achsen-Qualifizierer wird aus folgendem Beispiel ersichtlich:

```
...
const
        enable = 1;
var
        input: boolean;
...
input := A2.digib.enable; // Digitaleingang 1 von Achskanal 2 (I1)
                                                    // einlesen
A1.digob.7 := TRUE;       // Digitalausgang 7 (O7) setzen
...
```

Tabelle 38: Strukturierte Achsen-Qualifizierer

Name	Typ	Kurzwortbedeutung	Funktion
digib	boolean	digital-input-bit	Digital-Eingänge der APCI-800x (bitweise)
digob	boolean	digital-output-bit	Digital-Ausgänge der APCI-800x (bitweise)
ifsb	boolean	interface-status-bit	Status-Flags der APCI-800x (bitweise)
axstb	boolean	axis status-bit	Error-, Status- und Profil-Flags (bitweise)

Die Funktion dieser Qualifizierer kann bei den entsprechenden *rdxxxxb()*- bzw *wrxxxxb()*-Befehlen in der Funktionenreferenzliste der PCAP-Programmierung nachgelesen werden. Beispielsweise wird die Bedeutung des Qualifizierers *digib* beim Befehl *rddigib()* erklärt.

Anmerkung: Die Bit-Zählweise bei den strukturierten Achsenqualifizieren beginnt bei 1!

6.3.5 Abkürzungen

Vorab werden einige in der Funktionen-Referenzliste verwendete Abkürzungen erläutert:

Tabelle 39: Abkürzungen.

Name	Beschreibung
A1	Symbolischer Name für den ersten Achskanal. Dieser Name kann in mcfg.exe frei gewählt werden. Wird hauptsächlich bei Beispielen verwendet.
A2	Symbolischer Name für den zweiten Achskanal. Sonst wie A1.
Spec	Achsen-Spezifizierer wie z.B. A1 oder A2
Qual	Achsen-Qualifizierer wie z.B. digi, digib, digo, digob, axst usw.
Pos	Positionssollwert (Datentyp double)
Event	Prozedur mit Funktion als Eventhandler

6.4 Reservierte Prozedur-Namen mit Event-Funktion

In *rw_SymPas* sind eine Reihe von Prozedur-Namen mit Ereignis-Funktion vordefiniert. Sofern Prozedurdefinitionen mit diesen Prozedur-Namen im Anwenderprogramm erfolgen, kann die CNC-Task mit einem Freigabebefehl dazu veranlasst werden, diese Prozeduren beim Eintreffen eines prozedurspezifischen Ereignisses (Event) automatisch aufzurufen. Diese Prozeduren werden deshalb auch als Event-Handler bezeichnet.

Anmerkung: Die Events werden nach jeder Ausführung einer rw_SymPas-Anweisung abgeprüft. Hierbei ist unbedingt zu beachten, dass die entsprechenden Events nicht mehr überprüft werden, . wenn eine Task beendet ist. Wenn eine dauerhafte Eventüberwachung notwendig ist, muss die entsprechende Task in einer Endlosschleife verbleiben.

6.4.1 Event-Prozedur EVEO

Die Definition der Prozedur EVEO und Freigabe des entsprechenden Ereignisses bewirkt den automatischen Aufruf dieser Prozedur. Das Ereignis EO (Emergency Out. Noto-Aus) tritt dann auf, wenn ein mit EO-Funktion projektierter Digital-Eingang aktiviert wird (MCFG / Kapitel 1.7.2.5). Sofern mehrere EO-Eingänge im System vorhanden sind, kann mit dem Statusregister *axst* geprüft werden, welcher EO-Eingang den Fehler verursacht. Nachfolgend wird ein einfaches Beispielprogramm für die Implementierung eines EO-Handlers aufgelistet:

```
...
procedure EVEO;          // vordefinierter Name für
                         // Timeout-EVENT-Handler
begin
    CI0 := 999;          // Common Variable
                         // signalisiert Pogrammabbruch
    abort;               // Benutzerprogramm abbrechen
end;

...
begin
    ...
    CI0 := 0;            // Common Variable
                         // löschen
    enev(EVEO);          // Timeout-Handler freigeben
    ...
end.
```

6.4.2 Event-Prozedur EVDNR

Die Funktionsweise der Event-Prozedur EVDNR ist ebenfalls mit EVEO identisch, jedoch wird diese Prozedur beim Eintreffen des Ereignisses Drive Not Ready (Antrieb nicht bereit) automatisch abgearbeitet. Das Ereignis DNR tritt dann auf, wenn ein mit DR-Funktion projektierter Digital-Eingang inaktiv wird (MCFG / Kapitel 1.7.2.5).

6.4.3 Event-Prozedur EVLSH

Die Funktionsweise der Event-Prozedur EVLSH ist ebenfalls mit EVEO identisch, jedoch wird diese Prozedur beim Eintreffen des Ereignisses Limit Switch Hardware (Hardware-Endschalter) automatisch abgearbeitet. Das Ereignis LSH tritt dann auf, wenn ein mit LSL_SMD, LSL_TOM, LSL_SMA, LSR_SMD, LSR_TOM oder LSR_SMA-Funktion projektierter Digital-Eingang aktiviert wird (MCFG / Kapitel 1.7.2.5).

6.4.4 Event-Prozedur EVLSS

Die Funktionsweise der Event-Prozedur EVLSS ist ebenfalls mit EVEO identisch, jedoch wird diese Prozedur beim Eintreffen des Ereignisses Limit Switch Software (Software-Endlage) automatisch abgearbeitet. Das Ereignis LSS tritt dann auf, wenn die aktuelle Position eines Achssystems einen im TOOLSET-Programm *mcfg.exe* spezifizierten Grenzwert überschreitet und der entsprechende Grenzwert mit der Funktion TOM oder SMA projektiert wurde (MCFG / Kapitel 1.7.2.5).

6.4.5 Event-Prozedur EVMPE

Die Funktionsweise der Event-Prozedur EVMPE ist ebenfalls mit EVEO identisch, jedoch wird diese Prozedur beim Eintreffen des Ereignisses Maximum Position Error (Maximaler Schleppfehler) automatisch abgearbeitet. Das Ereignis MPE tritt dann auf, wenn der Regelkreis geschlossen ist und die Differenz von Soll- und Ist-Position eines Achssystems den im TOOLSET-Programm *mcfg.exe* spezifizierten Grenzwert überschreitet (MCFG / Kapitel 1.7.2.1.9).

6.4.6 Event-Prozedur EVUI

Die Funktionsweise der Event-Prozedur EVUI ist ebenfalls mit EVEO identisch, jedoch wird diese Prozedur beim Eintreffen des Ereignisses User Input automatisch abgearbeitet. Das Ereignis UI tritt dann auf, wenn ein mit UI projektierter Digital-Eingang aktiviert wird (MCFG / Kapitel 1.7.2.5). Der Benutzer hat die Möglichkeit mit UI-projektierten Digital-Eingängen benutzerspezifische Spezial-Funktionen im SAP-Programm aufzubauen. Das alternative zyklische Abfragen (Polling) kann entfallen.

6.4.7 Priorität und Abarbeitungsreihenfolge der Event-Prozeduren

Es ist möglich, dass verschiedene Ereignisse zum gleichen Zeitpunkt eintreffen. In diesem Fall ist die Priorität wie folgt gegeben:

Name der Prozedur	Priorität
EVEO	höchste Priorität
EVDNR	↓
EVLSH	
EVLSS	
EVMPE	
EVUI	niedrigste Priorität

Sofern momentan eine Event-Prozedur (Event 1) in Bearbeitung ist, wird das Eintreffen eines anderen Events (Event 2) mit niederer oder höherer Priorität ignoriert und erst nach Abarbeitung des momentanen Eventhandlers (Event 1) durchgeführt. Event 2 muss dann aber noch aktiv sein!

Anmerkung: Nach den *STOP* und *ABORT* SAP-Befehlen und während der Ausführung des *WT()* SAP-Befehls werden keine Event-Handler abgearbeitet.

6.5 SAP-Satz-Befehle

In nachfolgend aufgelisteter Befehls-Referenzliste sind eine Reihe von Befehlen enthalten, mit denen ein Programmaufbau mit Satzstruktur erzielt werden kann. Dies sind alle Befehle, deren Befehlsname mit dem Zeichen 'W' endet. Dazu gehören beispielsweise die SAP-Befehle *MLAW()*, *JAW()* oder *SSMSW()*. Diese Befehle warten automatisch das Profilende aller beteiligten Achsen ab, d.h. die nächste Anweisung wird erst beim Erreichen der Zielpositionen der angewählten Achsen abgearbeitet. Dazu prüft die CNC-Task zyklisch die Profil-Ende-Flags dieser Achsen ab und setzt das Programm ggf. bei der nächsten Anweisung fort. Bei diesem Abprüfen werden auch die oben freigegebenen EVENT-Handler berücksichtigt und im Bedarfsfall automatisch abgearbeitet.
Anmerkung: Eine andere Möglichkeit der Profilende-Abprüfung ist das Auswerten des *axst*-Achsenqualifizierers.

6.6 SAP-Befehls-Referenzliste *rw_SymPas*

6.6.1 Aufbau der Referenzliste

Die Referenzliste ist wie folgt aufgebaut:

KURZWORTBEDEUTUNG, BESCHREIBUNG	Dies ist der Name, mit dessen Hilfe die nachfolgend beschriebene Funktion aufgerufen wird. Hier steht die ausführliche Beschreibung des entsprechenden Funktionsnamens.
FUNKTIONSPARAMETER:	Sofern die Funktion eine Parameterübergabe verlangt, werden diese hier aufgeführt.
SYSTEMPARAMETER:	Verschiedene Funktionen werden unter Berücksichtigung verschiedener Systemparameter ausgeführt. Diese werden hier aufgeführt.
SIMULTANFUNKTION:	Bei verschiedenen Funktion ist es gestattet, eine oder mehrere Achsen zu spezifizieren, für welche die entsprechende Funktion auszuführen ist.
REFERENZEN:	Verweise auf andere Funktionen und Kapitel.
DEKLARATION:	Die formale Deklaration von vordefinierten Systemfunktionen, benutzerdefinierte Elemente sind kursiv gesetzt.
ERGEBNISTYP:	Der Typ des zurückgelieferten Wertes (nur bei Systemfunktionen).
BESCHREIBUNG	Klartextbeschreibung des Befehls.
ANMERKUNG:	Immer wiederkehrende Anmerkungen und Erläuterungen verweisen hier auf die entsprechenden Kapitel.
BEISPIEL:	Ein Beispiel für die entsprechende Funktion.

6.6.2 ABORT, abort

BESCHREIBUNG:	Dieser Befehl bewirkt das Abbrechen eines laufenden SAP-Programmes. Im Gegensatz zur *STOP*-Anweisung kann das Programm nicht mit dem PCAP-Befehl *contcnct()* bzw. SAP-Befehl *CONTCNCT()* fortgesetzt werden. Dies ist nur durch den PCAP-Befehl *startcnct()* bzw. PCAP-Befehl *STARTCNCT()* möglich.
ANMERKUNG:	Nach Ausführen des Befehls werden die freigegebenen EVENT-Handler-Prozeduren nicht mehr abgearbeitet.
BEISPIEL:	*ABORT;*

6.6.3 ABS, absolute function

BESCHREIBUNG:	Die Funktion liefert den absoluten Wert von *value* zurück.
DEKLARATION:	abs(*value*:double)
ERGEBNISTYP:	double
BEISPIEL:	*...* *var* *d1, d2: double;* *...* *d1 := -5.0;* *d2 := ABS(d1); // d2 := 5.0*

6.6.4 ACOS, arc cosine function

BESCHREIBUNG:	Die Funktion liefert den Arcuscosinus von *value* zurück. Das Argument *Value* muss im Bereich [-1..+1] liegen. Der Rückgabewert hat die Einheit Rad und liegt in den Grenzen [0..pi].
DEKLARATION:	acos(*value*:double)
ERGEBNISTYP:	double

6.6.5 ASIN, arc sine function

BESCHREIBUNG:	Die Funktion liefert den Arcussinus von *value* zurück. Das Argument *Value* muss im Bereich [-1..+1] liegen. Der Rückgabewert hat die Einheit Rad und liegt in den Grenzen [-pi/2..+pi/2].
DEKLARATION:	asin(*value*:double)
ERGEBNISTYP:	double

6.6.6 ATAN, arc tangent function

BESCHREIBUNG:	Die Funktion liefert den Arcustangens von *value* zurück. Der Rückgabewert hat die Einheit Rad und liegt in den Grenzen [-pi/2..+pi/2].
DEKLARATION:	atan(*value*:double)
ERGEBNISTYP:	double

6.6.7 AZO, activate zero offsets

BESCHREIBUNG:	PCAP-Befehl *azo()*
FUNKTIONSPARAMETER:	Integer-Konstante im Wertebereich 0..4
BEISPIEL:	*const Offsets1 = 1;* *azo(Offsets1); // Nullpunktverschiebungen Satz 1 aktivieren*

6.6.8 CL, close loop

BESCHREIBUNG:	PCAP-Befehl *cl()* [Kapitel 4.4.6]
FUNKTIONSPARAMETER:	*Spec*
SIMULTANFUNKTION:	ja
BEISPIEL:	*CL(A1, A2); // Achskanäle 1 und 2 in Lageregelung bringen*

6.6.9 CLV

BESCHREIBUNG:	PCAP-Befehl *clv()* [Kapitel 4.4.9]
FUNKTIONSPARAMETER:	*Spec*
SIMULTANFUNKTION:	ja

BEISPIEL:	*clv (A1, A2); // Achskanäle 1 und 2 in Lageregelung bringen*
	js (A1, A2); // danach die Achsen sofort stoppen

6.6.10 CONTCNCT, continue CNC-Task

BESCHREIBUNG:	Dieser Befehl setzt die im Parameter übergebene CNC-Task fort.
FUNKTIONSPARAMETER:	Integer-Konstante im Bereich 0..3
ANMERKUNG:	Der Befehl kann benutzt werden, um ein gestopptes SAP-Programm fortzusetzen. Ein SAP-Programm welches mit dem SAP-Befehl *ABORT* angehalten wurde, kann nur mit dem SAP-Befehl *STARTCNCT()* oder PCAP-Befehl *startcnct()* neu gestartet, also nicht fortgesetzt, werden. Das selbsttätige Fortsetzen einer gestoppten Task ist ebenfalls nicht möglich.

BEISPIEL:	*...*
	const
	TASK0 = 0;
	...
	CONTCNCT(TASK0); // Task 0 fortsetzen
	CONTCNCT(1); // Task 1 fortsetzen

6.6.11 COS, cosine function

BESCHREIBUNG:	Die Funktion liefert den Cosinus von *value* zurück. Das Argument *Value* wird als Winkel in der Einheit Rad (0..2Pi = 0..360 Grad) interpretiert.
DEKLARATION:	cos(*value*:double)
ERGEBNISTYP:	double
ANMERKUNG:	*Sin()*, *Tan()*-Funktion
BEISPIEL:	*...* *var* *d1, d2: double;* *...* *d1 := 3.1415;* *d2 := COS(d1); // d2 := -1.0 (gerundet)*

6.6.12 COSH, hyperbolic cosine function

BESCHREIBUNG:	Die Funktion liefert den hyperbolischen Cosinus von *value* zurück.
DEKLARATION:	cos(*value*:double)
ERGEBNISTYP:	double

6.6.13 DISEV, disable event

BESCHREIBUNG:	sperrt den spezifizierten Event-Handler
FUNKTIONSPARAMETER:	*Event*
REFERENZEN:	Kapitel 6.4. und SAP-Befehl *ENEV()*
BEISPIEL:	*DISEV(EVEO); // Emergency Out Handler ignorieren*

6.6.14 ENEV, enable event

BESCHREIBUNG:	gibt den spezifizierten Event-Handler frei
FUNKTIONSPARAMETER:	*Event*
REFERENZEN:	Kapitel 6.4. und SAP-Befehl *DISEV()*
BEISPIEL:	*ENEV(EVEO); // Emergency Out Handler freigeben*
ANMERKUNG:	Der freigegebene Event-Handler ist nicht mehr aktiv, wenn die Task beendet ist bzw. angehalten wurde.

6.6.15 EXP, exponential function

BESCHREIBUNG:	Die Funktion liefert den Wert e^{value} zurück, wobei *e* die Basis des natürlichen Logarithmus ist (2.718281...).
DEKLARATION:	exp(*value*:double)
ERGEBNISTYP:	double
ANMERKUNG:	Funktion *Ln()*

6.6.16 JA, jog absolute

BESCHREIBUNG:	Der oder die ausgewählten Achskanäle werden auf die angegebenen Positionssollwerte absolut verfahren. Dazu wird der Motor mit der achsspezifischen Beschleunigung *jac* auf die Geschwindigkeit *jvl* hochbeschleunigt und auf die spezifizierte Zielposition *Pos* verfahren. Zusätzlich kann mit dem Parameter *jtl* eine Zielgeschwindigkeit spezifiziert werden. Die Angabe der Bahnparameter erfolgt in den achsenspezifischen Einheiten.
FUNKTIONSPARAMETER:	*Spec* und *Pos*
SYSTEMPARAMETER:	Qualifizierer: *jac, jvl* und *jtvl*
SIMULTANFUNKTION:	ja
REFERENZEN:	PCAP-Befehl *ja()*, SAP-Befehl *JAW()*
ANMERKUNG:	PCAP-Befehl *ja()*

BEISPIEL:	*JA(A1:=100.0); // Achse 1 absolut auf Position 100 verfahren*
	JA(A1:=100.0, A2:=100.0);

6.6.17 JAW, jog absolute waiting

BESCHREIBUNG:	Dieser Befehl ist mit dem SAP-Befehl *JA()* und PCAP-Befehl *ja()* identisch, wartet jedoch zusätzlich das Profilende aller beteiligten Achsen ab. Durch den Einsatz dieses Befehls erhält das SAP-Programm eine satzähnliche Gestalt wie sie bei den handelsüblichen CNC-Steuerungen zu finden ist.
FUNKTIONSPARAMETER:	*Spec, Pos*
SYSTEMPARAMETER:	Qualifizierer: *jac, jvl* und *jtvl*
SIMULTANFUNKTION:	ja
REFERENZEN:	*JA*
ANMERKUNG:	Der Benutzer sollte durch den Einsatz von EVENT-Handlern sicherstellen, dass der Antrieb auch in Ausnahmesituationen korrekt bedient wird, da das CNC-Programm gerade bei Positioniervorgängen mit großem Zeitbedarf entsprechend lange bei diesem Befehl verweilt.

BEISPIEL:	*JAW(A2:=-1000.0); // Achse 1 absolut auf Position -1000.0 verfahren*
	// und warten bis das Profilende erreicht wird
	JAW(A1:=1e3, A2 := 1.3e4);

6.6.18 JHI, jog home index

BESCHREIBUNG:	Der Referenzsuchlauf auf die Nullspur (Index) des Drehgebers oder des Linearmaßstabes aller selektierten Achskanäle wird gestartet. Der Suchlauf wird abgebrochen, sobald der in *Pos* spezifizierte Verfahrweg bzw. Winkel überschritten wird.
FUNKTIONSPARAMETER:	*Spec, Pos*
SYSTEMPARAMETER:	Qualifizierer: *hac* und *hvl*
SIMULTANFUNKTION:	ja
REFERENZEN:	PCAP-Befehl *jhi()*, SAP-Befehl *JHIW()*

BEISPIEL:	*JHI(A1 := 1.0, A2 := 1.5); // Referenzsuchlauf von Achse 1 und 2*
	// starten.

6.6.19 JHIW, jog home index waiting

BESCHREIBUNG:	Dieser Befehl ist identisch mit dem PCAP-Befehl *jhi()* und SAP-Befehl *JHI()*. Zusätzlich wird das Profilende der beteiligten Achsen abgewartet.
FUNKTIONSPARAMETER:	*Spec*
SYSTEMPARAMETER:	Qualifizierer: *hac* und *hvl*
SIMULTANFUNKTION:	ja
ANMERKUNG:	SAP-Befehl *JA()*
BEISPIEL:	*JHIW(A1 := 5.0);*

6.6.20 JHL, jog home left

BESCHREIBUNG:	Der Referenzsuchlauf auf einen mit REF projektierten Digitaleingang aller selektierten Achskanäle wird in die linke Verfahrrichtung gestartet.
FUNKTIONSPARAMETER:	*Spec*
SYSTEMPARAMETER:	Qualifizierer: *hac* und *hvl*
SIMULTANFUNKTION:	ja
REFERENZEN:	PCAP-Befehl *jhl()*, SAP-Befehl *JHLW()*
BEISPIEL:	*JHL(A1);*

6.6.21 JHLW, jog home left waiting

BESCHREIBUNG:	Dieser Befehl ist identisch mit dem PCAP-Befehl *jhl()* und SAP-Befehl *JHL()*. Zusätzlich wird das Profilende der beteiligten Achsen abgewartet.
FUNKTIONSPARAMETER:	*Spec*
SYSTEMPARAMETER:	Qualifizierer: *hac* und *hvl*
SIMULTANFUNKTION:	ja
REFERENZEN:	PCAP-Befehl *jhl()*, SAP-Befehl *JHL()*
ANMERKUNG:	SAP-Befehl *JA()*
BEISPIEL:	*JHLW(A2);*

6.6.22 JHR, jog home right

BESCHREIBUNG:	Der Referenzsuchlauf auf einen mit REF projektierten Digitaleingang aller selektierten Achskanäle wird in die rechte Verfahrrichtung gestartet.
FUNKTIONSPARAMETER:	*Spec*
SYSTEMPARAMETER:	Qualifizierer: *hac* und *hvl*
SIMULTANFUNKTION:	ja
REFERENZEN:	PCAP-Befehl *jhr()*, SAP-Befehl *JHRW()*
BEISPIEL:	*JHR(A2);*

6.6.23 JHRW, jog home right waiting

BESCHREIBUNG:	Dieser Befehl ist identisch mit dem PCAP-Befehl *jhr()* und SAP-Befehl *JHR()*. Zusätzlich wird das Profilende der beteiligten Achsen abgewartet.
FUNKTIONSPARAMETER:	*Spec*
SYSTEMPARAMETER:	Qualifizierer: *hac* und *hvl*
SIMULTANFUNKTION:	ja
ANMERKUNG:	SAP-Befehl *JA()*
BEISPIEL:	*JHRW(A1);*

6.6.24 JR, jog relative

BESCHREIBUNG:	Die Beschreibung erfolgt beim PCAP-Befehl *jr()*.
FUNKTIONSPARAMETER:	*Spec, Po*s
SYSTEMPARAMETER:	Qualifizierer: *jac, jvl* und *jtvl*
SIMULTANFUNKTION:	ja
BEISPIEL:	*JR(A1 := 100);*

6.6.25 JRW, jog relative waiting

BESCHREIBUNG:	Dieser Befehl ist identisch mit dem PCAP-Befehl *jr()* und SAP-Befehl *JR()*. Zusätzlich wird das Profilende der beteiligten Achsen abgewartet.
FUNKTIONSPARAMETER:	*Spec, Po*s
SYSTEMPARAMETER:	Qualifizierer: *jac, jvl* und *jtvl*
SIMULTANFUNKTION:	ja
REFERENZEN:	JR

6.6.26 JS, jog stop

BESCHREIBUNG:	Die Beschreibung erfolgt beim PCAP-Befehl *js()*.
FUNKTIONSPARAMETER:	*Spec*
SYSTEMPARAMETER:	Qualifizierer: *sdec*
SIMULTANFUNKTION:	ja
BEISPIEL:	*JS(A1);*

6.6.27 JSW, jog stop waiting

BESCHREIBUNG:	Dieser Befehl ist identisch mit dem PCAP-Befehl *js()* und SAP-Befehl *JS()*. Zusätzlich wird das Profilende der beteiligten Achsen abgewartet.
FUNKTIONSPARAMETER:	*Spec*
SYSTEMPARAMETER:	Qualifizierer: *sdec*
SIMULTANFUNKTION:	ja
BEISPIEL:	*JSW(A1);*

6.6.28 LN, natural logarithm function

BESCHREIBUNG:	Die Funktion liefert den natürlichen Logarithmus von *value* zurück, d.h. der Wert, mit dem die Konstante 2.71828... potenziert werden muss, um *value* zu erhalten.
DEKLARATION:	ln(*value*:double)
ERGEBNISTYP:	double
ANMERKUNG:	Wert kleiner/gleich 0.0 für *value* sind mathematisch nicht definiert. Die Funktion hat in diesem Fall keinen gültigen Rückgabewert. Funktion *Exp()*

6.6.29 LPR, latch position registers

BESCHREIBUNG:	Daten-Aufzeichnung eines Bewegungsvorganges für eine Achse starten (siehe grafische Systemanalyse in mcfg).
FUNKTIONSPARAMETER:	*Spec*
SYSTEMPARAMETER:	PU, TU, LST, LET
SIMULTANFUNKTION:	nein
BEISPIEL:	LPR (A1);

6.6.30 LPRS, latch position registers synchronous

BESCHREIBUNG:	Synchrone Daten-Aufzeichnung eines Bewegungsvorganges für mehrere Achsen starten (siehe grafische Systemanalyse in mcfg).
FUNKTIONSPARAMETER:	*Spec*
SYSTEMPARAMETER:	PU, TU, LST, LET
SIMULTANFUNKTION:	ja
BEISPIEL:	LPRS (A1, A2, A3);

6.6.31 MCA, move circular absolute
SMCA, spool motion circular absolute

BESCHREIBUNG:	PCAP-Befehl *mca()*, *smca()*
FUNKTIONSPARAMETER:	*Spec*, *Pos*
SYSTEMPARAMETER:	*TRAC, TRVL, TRTVL, PHI*
BEISPIEL:	*MCA(A1 := 50.0, A2 := 0.0, PHI := 720.0);* *SMCA(A1 := 0.0, A2 := 10.0, PHI := 0.1);*

6.6.32 MCAW, move circular absolute waiting

BESCHREIBUNG:	Dieser Befehl ist identisch mit dem SAP-Befehl *MCA()*, jedoch wird hier zusätzlich das Profilende der beiden beteiligten Achsen abgewartet.
FUNKTIONSPARAMETER:	*Spec*, *Pos*
SYSTEMPARAMETER:	*TRAC, TRVL, TRTVL, PHI*
REFERENZEN:	PCAP-Befehl *mca()*

6.6.33 MCA3D, move circular absolute three-dimendional
SMCA3D, spool move circular absolute three-dimendional

BESCHREIBUNG:	PCAP-Befehl *mca3d()*, *smca3d()*
FUNKTIONSPARAMETER:	*Spec*, *Pos*
SYSTEMPARAMETER:	*TRAC, TRVL, TRTVL, PHI, PN1, PN2, PN3*
BEISPIEL:	*MCA3D(A1 := 50.0, A2 := 0.0, A3 := 0.0, PN1 = 1.0, PN2 =0. 0, PN3 = 1.0, PHI := 720.0); // Kreis 45 Grad um A2 gedreht*

6.6.34 MCA3DW, move circular absolute three-dimendional waiting

BESCHREIBUNG:	Dieser Befehl ist identisch mit dem SAP-Befehl *MCA3D()*, jedoch wird hier zusätzlich das Profilende der beiden beteiligten Achsen abgewartet.
FUNKTIONSPARAMETER:	*Spec*, *Pos*
SYSTEMPARAMETER:	*TRAC, TRVL, TRTVL, PHI, PN1, PN2, PN3*
REFERENZEN:	SAP-Befehl *mca3d()*

6.6.35 MCR3D, move circular relative three-dimendional
SMCR3D, spool move circular relative three-dimendional

BESCHREIBUNG:	PCAP-Befehl *mcr3d()*, *smcr3d()*
FUNKTIONSPARAMETER:	*Spec*, *Pos*
SYSTEMPARAMETER:	*TRAC, TRVL, TRTVL, PHI, PN1, PN2, PN3*
BEISPIEL:	*MCR3D(A1 := 50.0, A2 := 0.0, A3 := 0.0, PN1 = 1.0, PN2 =0. 0, PN3 = 1.0, PHI := 720.0); // Kreis 45 Grad um A2 gedreht*

6.6.36 MCR3DW, move circular relative three-dimendional waiting

BESCHREIBUNG:	Dieser Befehl ist identisch mit dem SAP-Befehl *MCR3D()*, jedoch wird hier zusätzlich das Profilende der beiden beteiligten Achsen abgewartet.
FUNKTIONSPARAMETER:	*Spec*, *Pos*
SYSTEMPARAMETER:	*TRAC, TRVL, TRTVL, PHI, PN1, PN2, PN3*
REFERENZEN:	SAP-Befehl *mcr3d()*

6.6.37 MCR, move circular relative
SMCR, spool motion circular relative

BESCHREIBUNG:	PCAP-Befehl *mcr()*, *smcr()*
FUNKTIONSPARAMETER:	*Spec*, *Pos*
SYSTEMPARAMETER:	*TRAC, TRVL, TRTVL, PHI*
BEISPIEL:	*MCR(A1 := 50.0, A2 := 0.0, PHI := 360.0);* *SMCR(A1 := 0.0, A2 := 10.0, PHI := 45.0);*

6.6.38 MCRW, move circular relative waiting

BESCHREIBUNG:	Dieser Befehl ist identisch mit dem SAP-Befehl *MCR()*, jedoch wird hier zusätzlich das Profilende der beiden beteiligten Achsen abgewartet.
FUNKTIONSPARAMETER:	*Spec*, *Pos*
SYSTEMPARAMETER:	*TRAC, TRVL, TRTVL, PHI*

6.6.39 MHA, move helical absolute
SMHA, spool motion helical absolute

BESCHREIBUNG:	PCAP-Befehl *mha()*, *smha()* *Falls der Kreis hier per Zielpunktsangabe definiert werden soll, müssen die Zielkoordinaten den* Achsenspezifizierern*, die Mittelpunktskoordinaten den Systemparametern DTCA1 und DTCA2 zugewiesen werden. Bei Kreisangabe per Verfahrwinkel werden den* Achsenspezifizierern der Kreisachsen die Mittelpunktskoordinaten zugewiesen.
FUNKTIONSPARAMETER:	*Spec*, *Pos*
SYSTEMPARAMETER:	*TRAC, TRVL, TRTVL, PHI, (ggf. DTCA1, DTCA2)*
BEISPIELE:	*MHA(A1 := 50.0, A2 := 0.0, PHI := 720.0, A3 := 10);* *SMHA(A1 := 0.0, A2 := 10.0, PHI := 0.1, A3 := 10);* *// Vollkreis im Gegenuhrzeigersinn mit Radius 10* *MHR(A1 := 0.0, A2 := 0.0, PHI := 0.0, A3 := 10, DTCA1 :=-10, DTCA2 := 0);* *// Halbkreis im Uhrzeigersinn mit Radius 10* *SMHR(A1 := 20.0, A2 := 0.0, PHI := -1e-100, A3 := 10, DTCA1 := 10, DTCA2 := 0);*

6.6.40 MHAW, move helical absolute waiting

BESCHREIBUNG:	Dieser Befehl ist identisch mit dem SAP-Befehl *MHA()*, jedoch wird hier zusätzlich das Profilende aller beteiligten Achsen abgewartet.
FUNKTIONSPARAMETER:	*Spec*, *Pos*
SYSTEMPARAMETER:	*TRAC, TRVL, TRTVL, PHI, (ggf. DTCA1, DTCA2)*

6.6.41 MHR, move helical relative
SMHR, spool motion helical relative

BESCHREIBUNG:	PCAP-Befehl *mhr()*, *smhr()* *Eine Zielpunktsangabe für den Kreis ist hier nicht möglich.*
FUNKTIONSPARAMETER:	*Spec*, *Pos*
SYSTEMPARAMETER:	*TRAC, TRVL, TRTVL, PHI, (ggf. DTCA1, DTCA2)*

6.6.42 MHRW, move helical relative waiting

BESCHREIBUNG:	Dieser Befehl ist identisch mit dem SAP-Befehl *MHR()*, jedoch wird hier zusätzlich das Profilende aller beteiligten Achsen abgewartet.
FUNKTIONSPARAMETER:	*Spec*, *Pos*
SYSTEMPARAMETER:	*TRAC, TRVL, TRTVL, PHI, (ggf. DTCA1, DTCA2)*

6.6.43 MLA, move linear absolute
SMLA, spool motion linear absolute

BESCHREIBUNG:	Die Beschreibung erfolgt bei dem PCAP-Befehl *mla()* bzw. *smla()*.
FUNKTIONSPARAMETER:	*Spec*, *Pos*
SYSTEMPARAMETER:	*TRAC, TRVL, TRTVL*
SIMULTANFUNKTION:	ja
BEISPIEL:	*MLA(A1:=1000.0, A2:=3.2e2);* *SMLA(A1:=100.0, A2:=-335.0);*

6.6.44 MLAW, move linear absolute waiting

BESCHREIBUNG:	Dieser Befehl ist identisch mit dem SAP-Befehl *MLA()*, jedoch wird hier zusätzlich das Profilende der beteiligten Achsen abgewartet.
FUNKTIONSPARAMETER:	*Spec*, *Pos*
SYSTEMPARAMETER:	*TRAC, TRVL, TRTVL*
SIMULTANFUNKTION:	ja
BEISPIEL:	*MLAW(A1:=-0.3e3, A2:=100.4);*

6.6.45 MLR, move linear relative
SMLR, spool motion linear relative

BESCHREIBUNG:	Die Beschreibung erfolgt bei dem PCAP-Befehl *mlr()* bzw. *smlr()*.
FUNKTIONSPARAMETER:	*Spec*, *Pos*
SYSTEMPARAMETER:	*TRAC, TRVL, TRTVL*
SIMULTANFUNKTION:	ja
BEISPIEL:	*MLR(A1:=2000.0, A2:=3.2e2);* *SMLR(A1:=300.0, A2:=-35.3);*

6.6.46 MLRW, move linear relative waiting

BESCHREIBUNG:	Dieser Befehl ist identisch mit dem SAP-Befehl *MLRW()*, jedoch wird hier zusätzlich das Profilende der beteiligten Achsen abgewartet.
FUNKTIONSPARAMETER:	*Spec*, *Pos*
SYSTEMPARAMETER:	*TRAC, TRVL, TRTVL*
SIMULTANFUNKTION:	ja
BEISPIEL:	*MLRW(A1:=-3.45e3, A2:=100.4e-1);*

6.6.47 MS, motion stop

BESCHREIBUNG:	Die Beschreibung erfolgt beim PCAP-Befehl *ms()* [Kapitel 4.4.39].
FUNKTIONSPARAMETER:	*Spec*
SYSTEMPARAMETER:	keine
SIMULTANFUNKTION:	ja
BEISPIEL:	*MS(A1, A2);*

6.6.48 MSW, motion stop waiting

BESCHREIBUNG:	Dieser Befehl ist identisch mit dem PCAP-Befehl *ms()* und SAP-Befehl *MS()*, jedoch wird hier zusätzlich das Profilende der beteiligten Achsen abgewartet.
FUNKTIONSPARAMETER:	*Spec*
SYSTEMPARAMETER:	keine
SIMULTANFUNKTION:	ja
BEISPIEL:	*MSW(A1, A2);*

6.6.49 OL, open loop

BESCHREIBUNG:	PCAP-Befehl *ol()* [Kapitel 4.4.41]
FUNKTIONSPARAMETER:	*Spec*
SIMULTANFUNKTION:	ja
BEISPIEL:	*OL(A1, A2); // Lageregelkreis von A1 und A2 öffnen*

6.6.50 POWER

BESCHREIBUNG:	Die Funktion liefert den Wert von *base* hoch *exponent* zurück.
DEKLARATION:	sqrt(*base, exponent* : double)
ERGEBNISTYP:	double
ANMERKUNG:	Funktion ist verfügbar ab RWMOS.ELF V2.5.3.93

BEISPIEL:	*...*
	var
	d1, d2: double;
	...
	d1 := 2.0;
	d2 := 3.0;
	d2 := POWER(d1, d2); // d2 := 8.0

6.6.51 RA, reset axis

BESCHREIBUNG:	PCAP-Befehl *ra()*
FUNKTIONSPARAMETER:	*Spec*
SIMULTANFUNKTION:	ja
BEISPIEL:	*RA(A1, A2); // Achsen A1 und A2 zurücksetzen*

6.6.52 RDCBD, read COMMON BUFFER double function

BESCHREIBUNG:	Die Funktion liefert einen Gleitpunktwert mit doppelter Genauigkeit (double) aus dem CNC-taskspezifischen COMMON BUFFER zurück. Der Parameter *offset* ist ein Byte-Offset bezogen auf das erste Element (Element 0) des COMMON BUFFER. Der Double-Datentyp belegt 8 Bytes im COMMON BUFFER. Um einen korrekten Zugriff durch das APCI-800x-CPU-System zu ermöglichen, muss *offset* immer doppelwortgerichtet sein, d.h. einen durch 8 teilbaren Wert haben.
DEKLARATION:	RDCBD(*offset*:integer)
ERGEBNISTYP:	double
ANMERKUNG:	Die CNC-Task-spezifische Buffergröße beträgt 1000 Bytes. PCAP-Befehle *rdcbcnct() und wrcbcnct()*, SAP-Befehle *RDCBx()* und *WRCBx()*
BEISPIEL:	*...* *var* *cbd: double;* *...* *cbd := RDCBD(500); // Double-Variable einlesen ab offset 500*

6.6.53 RDCBI, read COMMON BUFFER integer function

BESCHREIBUNG:	Die Funktion liefert einen Integerwert aus dem CNC-taskspezifischen COMMON BUFFER zurück. Der Parameter *offset* ist ein Byte-Offset bezogen auf das erste Element (Element 0) des COMMON BUFFER. Der Integer-Datentyp belegt 4 Bytes im COMMON BUFFER. Um einen korrekten Zugriff durch das APCI-800x-CPU-System zu ermöglichen, muss *offset* immer wortgerichtet sein, d.h. einen durch 4 teilbaren Wert haben.
DEKLARATION:	RDCBI(*offset*:integer)
ERGEBNISTYP:	integer
ANMERKUNG:	Die CNC-Task-spezifische Buffergröße beträgt 1000 Bytes. PCAP-Befehle *rdcbcnct() und wrcbcnct()*, SAP-Befehle *RDCBx()* und *WRCBx()*.
BEISPIEL:	*...* *var* *cbi: integer;* *...* *cbi := RDCBI(500); // Integer-Variable einlesen ab offset 500*

6.6.54 RDCBS, read COMMON BUFFER single function

BESCHREIBUNG:	Die Funktion liefert einen Gleitpunktwert mit einfacher Genauigkeit (single) aus dem CNC-taskspezifischen COMMON BUFFER zurück. Der Parameter *offset* ist ein Byte-Offset bezogen auf das erste Element (Element 0) des COMMON BUFFER. Der Single-Datentyp belegt 4 Bytes im COMMON BUFFER. Um einen korrekten Zugriff durch das APCI-800x-CPU-System zu ermöglichen, muss *offset* immer wortgerichtet sein, d.h. einen durch 4 teilbaren Wert haben.
DEKLARATION:	RDCBS(*offset*:integer)
ERGEBNISTYP:	single
ANMERKUNG:	Die CNC-Task-spezifische Buffergröße beträgt 1000 Bytes. PCAP-Befehle *rdcbcnct() und wrcbcnct()*, SAP-Befehle *RDCBx()* und *WRCBx()*
BEISPIEL:	*...* *var* *cbs: single;* *...* *cbs := RDCBS(500); // Single-Variable einlesen ab offset 500*

6.6.55 RPTODP, Real-Position to Desired-Position

BESCHREIBUNG:	PCAP-Befehl *RPtoDP()*
ANMERKUNG:	Die betreffenden Achsen dürfen sich nicht in einem Verfahrprofil befinden, d.h. das Profil-Ende-Flag im Achsenstatusregister muss gesetzt sein.
BEISPIEL:	*RPTODP (X, Z);*

6.6.56 RS, reset system

BESCHREIBUNG:	PCAP-Befehl *rs()*
ANMERKUNG:	Sofern dieser Befehl ausgeführt wird, ist keine Kontrolle durch das Standalone-Applikations-Programm mehr möglich, da die CNC-Task durch diesen Befehl angehalten wird.
BEISPIEL:	*RS*; *// komplettes Achssytem zurücksetzen*

6.6.57 SHP, set home position

BESCHREIBUNG:	PCAP-Befehl *shp()*
FUNKTIONSPARAMETER:	*Spec, Pos*
SIMULTANFUNKTION:	ja
BEISPIEL:	*SHP(A2:=1000.0);*

6.6.58 SIN, sine function

BESCHREIBUNG:	Die Funktion liefert den Sinus von *value* zurück. Das Argument *Value* wird als Winkel in der Einheit Rad (0..2Pi = 0..360 Grad) interpretiert.
DEKLARATION:	sin(*value*:double)
ERGEBNISTYP:	double
ANMERKUNG:	*Cos()*, *Tan()*-Funktion
BEISPIEL:	*...* *var* *d1, d2: double;* *...* *d1 := 3.1415;* *d2 := SIN(d1);* *// d2 := 0.0 (gerundet)*

6.6.59 SINH, hyperbolic sine function

BESCHREIBUNG:	Die Funktion liefert den hyperbolischen Sinus von *value* zurück.
DEKLARATION:	cos(*value*:double)
ERGEBNISTYP:	double

6.6.60 SQR, square function

BESCHREIBUNG:	Die Funktion liefert das Quadrat (»square«) von value zurück.
DEKLARATION:	sqr(*value*:double)
ERGEBNISTYP:	double
ANMERKUNG:	Verfügbar nur in RWMOS und Compilerversionen ab 05.10.2007

BEISPIEL:	*...*
	var
	d1, d2: double;
	...
	d1 := 9.0;
	d2 := SQR(d1); // d2 := 81.0

6.6.61 SQRT, square root function

BESCHREIBUNG:	Die Funktion liefert die Quadratwurzel (»square root«) von value zurück.
DEKLARATION:	sqrt(*value*:double)
ERGEBNISTYP:	double
ANMERKUNG:	Negative Werte von *value* sind mathematisch nicht definiert. Die Funktion hat in diesem Fall keinen gültigen Rückgabewert.

BEISPIEL:	*...*
	var
	d1, d2: double;
	...
	d1 := 9.0;
	d2 := SQRT(d1); // d2 := 3.0

6.6.62 SSF, Spool-Special-Function

BESCHREIBUNG:	Dieses Kommando ermöglicht dem Anwender auch andere Kommandos als Verfahrbefehle im Spooler einzutragen. In der Systemvariable *SSFP* wird das auszuführende Kommando eingetragen. Der Wert *Value* wird als Parameter bei der jeweiligen Achse eingetragen. Die verfügbaren Kommandos sind beim PCAP-Kommando ssf aufgeführt.
FUNKTIONSPARAMETER:	*Spec*, *Value*
SYSTEMPARAMETER:	*SSFP*
SIMULTANFUNKTION:	ja
KOMMANDOS:	siehe PCAP-Kommando ssf Kapitel 4.4.122

BEISPIEL:	SSF(A1:=999, SSFP = 1); // CI1 mit 999 beschreiben
	SSF(A1:=1, A4:=2, SSFP := 1001); // O1 bei Achse 1 und O2 bei Achse 4 // setzen
	SSF(A1:=0, A2:=0, A3:=0, SSFP:=1000); // Spoolerhalt bei A1, A2 und A3

6.6.63 SSMS, start spooled motions synchronous

BESCHREIBUNG:	Mit Hilfe von *spool*-Befehlen können Kommandos an die einzelnen Achskanäle der APCI-800x übertragen werden. Diese werden in einer Warteschlange eingetragen. Der Befehl *SSMS()* veranlasst den Synchronstart für die Spoolerbefehlsabarbeitung aller in *AS* spezifizierten Achsen.
FUNKTIONSPARAMETER:	*Spec*
SIMULTANFUNKTION:	ja
REFERENZEN:	PCAP-Befehl *ssms()*, SAP-Befehl *SSMSW()*

BEISPIEL:	*...*
	SMLA(A1:=1000.0, A2:=1000.0); // Verfahrbefehl spoolen
	SMLR(A1:=200.0, A2:=500.0); // Verfahrbefehl spoolen
	...
	SSMS(A1, A2); // Spooler starten

6.6.64 SSMSW, start spooled motions synchronous waiting

BESCHREIBUNG:	Synchronstart aller angewählten Achsen und abwarten, bis sämtliche gespoolten Bewegungsprofile dieser Achsen komplett abgefahren sind und das Profilende aller beteiligten Achsen erreicht wird.
FUNKTIONSPARAMETER:	*Spec*
SIMULTANFUNKTION:	ja
REFERENZEN:	SAP-Befehl *SSMS()*
ANMERKUNG:	SPOOL-Modus

BEISPIEL:	*...*
	SMLR(A1:=1000.0, A2:=1000.0); // Verfahrbefehl spoolen
	SMLR(A1:=200.0; A2:=500.0); // Verfahrbefehl spoolen
	...
	SSMSW(A1, A2); // Spooler starten

6.6.65 STARTCNCT, start CNC-Task

BESCHREIBUNG:	Dieser Befehl startet die im Parameter übergebene CNC-Task und führt das dort abgelegte SAP-Programm vom Programmbeginn an aus.
FUNKTIONSPARAMETER:	Integer-Konstante im Bereich 0..3
ANMERKUNG:	Ein SAP-Programm kann sich mit diesem Befehl auch selbsttätig vom Programmbeginn an starten.
BEISPIEL:	*...* *const* *Task1 = 1;* *...* *STARTCNCT(Task1);*

6.6.66 STOP, stop

BESCHREIBUNG:	Dieser Befehl bewirkt den Programmstop des momentan ablaufenden Standalone-Applikations-Programms. Zusätzlich wird die entsprechende CNC-Task (Task 0, 1, 2, oder 3) in den Idle-Zustand gebracht. Das Applikations-Programm kann mit Hilfe des *contcnct()*-PCAP-Befehls, des CONTCNCT()-SAP-Befehls oder im TOOLSET-Programm *mcfg.exe* wieder fortgesetzt werden.
ANMERKUNG:	Eventuell freigegebene EVENT-Handling-Prozeduren werden nach Ausführen des Stop-Befehls nicht mehr abgearbeitet. Der Antrieb sollte vor Ausführung dieses Befehls deshalb in einen sicheren Betriebszustand gebracht werden.
BEISPIEL:	*STOP; // Stoppt das SAP-Programm*

6.6.67 STEPCNCT, step CNC-Task

BESCHREIBUNG:	Dieser Befehl führt eine Programmzeile aus in der angegebenen CNC-Task.
FUNKTIONSPARAMETER:	Integer-Konstante im Bereich 0..3
ANMERKUNG:	Eventuell freigegebene EVENT-Handling-Prozeduren werden nach Ausführung der Programmzeile in der entsprechend selektierten Task nicht mehr abgearbeitet. Vor Ausführung dieses Befehls muss ein gültiges Programm geladen sein. Siehe dazu auch PCAP Kommando *stepcnct* (Kapitel 4.4.125).
BEISPIEL:	*...* *const* *Task3 = 3;* *...* *STEPCNCT(Task3);*

6.6.68 STOPCNCT, stop CNC-Task

BESCHREIBUNG:	Dieser Befehl hält die im Parameter übergebene CNC-Task an und damit auch das dort abgelegte SAP-Programm.
FUNKTIONSPARAMETER:	Integer-Konstante im Bereich 0..3
ANMERKUNG:	Eventuell freigegebene EVENT-Handling-Prozeduren werden nach Ausführung des *STOPCNCT()* von der entsprechend selektierten Task nicht mehr abgearbeitet. Siehe dazu auch Kapitel 6.6.67.
BEISPIEL:	*...* *const* *Task3 = 3;* *...* *STOPCNCT(Task3);*

6.6.69 STOPTOSS

BESCHREIBUNG:	Dieser Befehl versetzt die im Parameter übergebene CNC-Task vom Stop-Zustand in den Step-Zustand, ohne jedoch eine Programmzeile in der angegebenen Task auszuführen.
FUNKTIONSPARAMETER:	Integer-Konstante im Bereich 0..3
ANMERKUNG:	Wenn sich die angegebene Task nicht im Stop-Modus befindet hat das Kommando keinen Einfluss. Dieses Kommando wird vorzugsweise zur Single-Step Abarbeitung unter Verwendung mehrerer SAP Programmtasks benötigt. Eventuell zuvor gespoolte Verfahrkommandos werden bei den aktuell verwendeten Achsen ausgeführt.
BEISPIEL:	*...* *const* *Task3 = 3;* *...* *STOPTOSS(Task3);*

6.6.70 SZPA – Set Zero Position Absolut

BESCHREIBUNG:	Setzen eines absoluten virtuellen Nullpunktes. Dieses Kommando ist beim PCAP-Kommando szpa dokumentiert (Kapitel 4.4.127). Die Verwendung von SZPA bei Schrittmotorsystemen ist erst ab RWMOS.ELF Version 2.5.2.32 möglich.
FUNKTIONSPARAMETER:	*Spec, Pos*
SIMULTANFUNKTION:	ja
REFERENZEN:	PCAP-Befehle *szpa()*, szpr(), SAP-Befehl *SZPR*
BEISPIEL:	SZPA (X := 100, Y := -20);

6.6.71 SZPR – Set Zero Position Relativ

BESCHREIBUNG:	Relatives Verschieben des virtuellen Nullpunktes. Dieses Kommando ist beim PCAP-Kommando szpr dokumentiert (Kapitel 4.4.128). Die Verwendung von SZPR bei Schrittmotorsystemen ist erst ab RWMOS.ELF Version 2.5.2.32 möglich.
FUNKTIONSPARAMETER:	*Spec, Pos*
SIMULTANFUNKTION:	ja
REFERENZEN:	PCAP-Befehle *szpa()*, szpr(), SAP-Befehl *SZPA*
BEISPIEL:	SZPR (X := 100, Y := -20);

6.6.72 TAN, tangent function

BESCHREIBUNG:	Die Funktion liefert den Tangens von *value* zurück. Das Argument *Value* wird als Winkel in der Einheit Rad (0..2Pi = 0..360) Grad interpretiert.
DEKLARATION:	tan(*value*:double)
ERGEBNISTYP:	double
ANMERKUNG:	*Sin()*, *Cos()*-Funktion
BEISPIEL:	*...* *var* *d1, d2: double;* *...* *d1 := 0.5;* *d2 := TAN(d1); // d2 := 0.5463 (gerundet)*

6.6.73 TANH, hyperbolic tangent function

BESCHREIBUNG:	Die Funktion liefert den hyperbolischen Tangens von *value* zurück.
DEKLARATION:	tan(*value*:double)
ERGEBNISTYP:	double

6.6.74 UF, update filter

BESCHREIBUNG:	PCAP-Befehl *uf()*
FUNKTIONSPARAMETER:	*Spec*
SYSTEMPARAMETER:	Qualifizierer: *kp, ki, kd, kpl, kfca, kfcv*
SIMULTANFUNKTION:	ja
ANMERKUNG:	Zur Aktualisierung der PIDF-Filter-Koeffizienten müssen alle oben aufgeführten Qualifizierer vor Ausführung des Befehls initialisiert werden.

BEISPIEL:	*...*
	A1.kp := 5.0; // Proportionalverstärkung ändern
	A1.ki := 0.0;
	A1.kd := 0.0;
	A1.kpl := 0.0;
	A1.kfca := 0.0;
	A1.kfcv := 0.0;
	UF(A1);
	...

6.6.75 UTROVR, update trajectory override

BESCHREIBUNG:	PCAP-Befehl *utrovr()*
FUNKTIONSPARAMETER:	*Spec*
SYSTEMPARAMETER:	*TROVR*
SIMULTANFUNKTION:	ja

BEISPIEL:	*...*
	TROVR := 0.9; // Bahngeschwindigkeitsoverride = -10%
	UTROVR(A1, A2); // reduzierte Bahngeschwindigkeit für Achsen A1 und A2
	...

6.6.76 WRCBI, write COMMON BUFFER integer procedure

BESCHREIBUNG:	Die Prozedur beschreibt eine Speicherzelle vom Typ integer mit dem Wert *value* im CNC-taskspezifischen COMMON BUFFER. Der Parameter *offset* ist ein Byte-Offset bezogen auf das erste Element (Element 0) des COMMON BUFFER. Der Integer-Datentyp belegt 4 Bytes im COMMON BUFFER. Um einen korrekten Zugriff durch das APCI-800x-CPU-System zu ermöglichen, muss *offset* immer wortgerichtet sein, d.h. einen durch 4 teilbaren Wert haben.
DEKLARATION:	WRCBI(*offset*:integer; *value*:integer)
ANMERKUNG:	Die CNC-Task-spezifische Buffergröße beträgt 1000 Bytes. PCAP-Befehle *rdcbcnct() und wrcbcnct()*, SAP-Befehle *RDCBx()* und *WRCBx()*

BEISPIEL:	*WRCBI(500, -1000); // Integer-Variable beschreiben ab offset 500*
	// mit Wert -1000

6.6.77 WRCBS, write COMMON BUFFER single procedure

BESCHREIBUNG:	Die Prozedur beschreibt eine Speicherzelle vom Typ single (Gleitpunkt-Zahl mit einfacher Genauigkeit) mit dem Wert *value* im CNC-taskspezifischen COMMON BUFFER. Der Parameter *offset* ist ein Byte-Offset bezogen auf das erste Element (Element 0) des COMMON BUFFER. Der Single-Datentyp belegt 4 Bytes im COMMON BUFFER. Um einen korrekten Zugriff durch das APCI-800x-CPU-System zu ermöglichen, muss *offset* immer wortgerichtet sein, d.h. einen durch 4 teilbaren Wert haben.
DEKLARATION:	WRCBS(*offset*:integer; *value*:single)
ANMERKUNG:	Die CNC-Task-spezifische Buffergröße beträgt 1000 Bytes. PCAP-Befehle *rdcbcnct() und wrcbcnct()*, SAP-Befehle *RDCBx()* und *WRCBx().*

BEISPIEL:	*WRCBS(500, 3.99); // Single-Variable beschreiben ab offset 500*
	// mit Wert 3.99

6.6.78 WRCBD, write COMMON BUFFER double procedure

BESCHREIBUNG:	Die Prozedur beschreibt eine Speicherzelle vom Typ double (Gleitpunkt-Zahl mit doppelter Genauigkeit) mit dem Wert *value* im CNC-taskspezifischen COMMON BUFFER. Der Parameter *offset* ist ein Byte-Offset bezogen auf das erste Element (Element 0) des COMMON BUFFER. Der Double-Datentyp belegt 8 Bytes im COMMON BUFFER. Um einen korrekten Zugriff durch das APCI-800x-CPU-System zu ermöglichen, muss *offset* immer doppelwortgerichtet sein, d.h. einen durch 8 teilbaren Wert haben.
DEKLARATION:	WRCBD(*offset*:integer; *value*:double)
ANMERKUNG:	Die CNC-Task-spezifische Buffergröße beträgt 1000 Bytes. PCAP-Befehle *rdcbcnct() und wrcbcnct()*, SAP-Befehle *RDCBx()* und *WRCBx()*

BEISPIEL:	*WRCBD(500, 100.2e-128); // Double-Variable beschreiben ab offset 500*
	// mit Wert 100.2e-128

6.6.79 WRITE

BESCHREIBUNG:	Anhängen eines Teilstrings an die aktuelle taskspezifische Stringausgabe.
FUNKTIONSPARAMETER:	*diverse*
ANMERKUNG:	Die Funktion kann mit einer unbestimmten Anzahl von Parametern aufgerufen werden, die vom Typ Stringkonstante, Integer, Double oder Boolean sein können. Stringkonstanten sind Zeichenketten, die in rw_SymPas durch Hochkommata, in der G-Code Programmierung durch Anführungszeichen begrenzt sind. Die einzelnen Parameter werden durch Kommata getrennt. Numerische oder boolesche Parameter können auch Ausdrücke sein. Der Aufruf dieser Funktion setzt Bit 0 in der Systemvariablen tskinfo. Informationen über den Zustand der Stringausgabe siehe Kapitel 4.4.13. Das Lesen des Taskspezifisischen Ausgabestrings erfolgt mit der PCAP-Funktion gettskstr(), siehe Kapitel 4.4.14.
BEISPIEL RW_SYMPAS:	write ('Dies ist ein String: ', CI0); write ('Istposition: ', A1.rp);
BEISPIELG-CODES:	N0100 write “Dies ist ein String“, CI0

6.6.80 WRITELN

BESCHREIBUNG:	Anhängen eines Teilstrings an die aktuelle taskspezifische Stringausgabe und Abschließen des Ausgabestrings.
FUNKTIONSPARAMETER:	*diverse*
ANMERKUNG:	Die Funktion kann mit einer unbestimmten Anzahl von Parametern aufgerufen werden, die vom Typ Stringkonstante, Integer, Double oder Boolean sein können. Stringkonstanten sind Zeichenketten, die in rw_SymPas durch Hochkommata, in der G-Code Programmierung durch Anführungszeichen begrenzt sind. Die einzelnen Parameter werden durch Kommata getrennt. numerische oder boolesche Parameter können auch Ausdrücke sein. Wenn nach dieser Anweisung write oder writeln erneut aufgerufen wird, wird der bisherige Ausgabestring überschrieben. Der Aufruf dieser Funktion setzt Bit 1 in der Systemvariablen tskinfo. Informationen über den Zustand der Stringausgabe sind in Kapitel 4.4.13 zu finden Das Lesen des task-spezifischen Ausgabestrings erfolgt mit der PCAP-Funktion gettskstr(), siehe Kapitel 4.4.14. Wenn Bit 26 im Register MODEREG (Kapitel 6.3.1.5) gesetzt ist, bewirkt dieses Kommando einen Stopp der jeweiligen CNC-Task.
BEISPIEL RW_SYMPAS:	writeln (‘Dies ist ein String: ’, CI0); writeln (‘Istposition: ’, A1.rp);
BEISPIELG-CODES:	N0100 writeln “Dies ist ein String“, CI0

6.6.81 WT, wait timer

BESCHREIBUNG:	Die als Parameter übergebene Verweilzeit abwarten, bis das SAP-Programm wieder fortgesetzt wird. Dieser Befehl versetzt die CNC-Task in einen inaktiven Zustand und benötigt deshalb keine CPU-Zeit. Um das Master-CPU-System zu entlasten, kann dieser Befehl gegebenenfalls in Warteschleifen oder Ähnlichem eingesetzt werden.
FUNKTIONSPARAMETER:	Integer-Wert mit Einheit 64µs
ANMERKUNG:	Die EVENT-Handling-Prozeduren werden während der Ausführung dieses Befehls nicht abgearbeitet. Sollen diese jedoch überwacht werden, so kann dies beispielsweise durch mehrere *WT()*-Aufrufe mit kürzeren Verweilzeiten (evtl. in einer Schleife) erzwungen werden.
BEISPIEL:	*...* *CONST sec = 15625;* *...* *WT(5*sec); // 5s warten* *...*

6.7 Compiler-Befehle

Wie der Name vermuten lässt, weist ein Compiler-Befehl den Compiler während der Übersetzung eines Quelltextes an, bestimmte Operationen auszuführen (oder zu unterlassen). In *rw_SymPas* wird ein Compilerbefehl wie folgt aktiviert:

Innerhalb des SAP-Quelltextprogramms wird innerhalb eines Kommentars eine spezielle Syntax formuliert: Direkt auf die öffnende Klammer ({) folgen ein Dollarzeichen ($) und der Name des Befehls, der aus einem oder mehreren Buchstaben besteht. Diese »Kommentare« können - von einigen Ausnahmen abgesehen - an jeder Stelle des Quelltextes erscheinen, an der auch ein normaler Kommentar zulässig ist.

6.7.1 Include-Datei

BESCHREIBUNG:	Dieser Compilerbefehl weist den Compiler an, die durch *Dateiname* bezeichnete Datei einzulesen. Der Compiler verhält sich dabei im Prinzip so, als ob der gelesene Text anstelle des {$I}-Befehls stünde. *rw_SymPas* erlaubt die Verschachtelung von Include-Dateien bis zu 15 Ebenen. Eine via {$I} eingefügte Datei kann also ihrerseits weitere Dateien einfügen, die wiederum {$I}-Befehle enthalten. **Anmerkung:** Sofern in der *mcfg.exe* NCC-Editor-Umgebung bereits eine Include-Datei in einem der drei Editor-Fenster eröffnet wurde, wird der SAP-Quelltext dieses Editors - und nicht der Inhalt der entsprechenden Datei - eingebunden.
SYNTAX:	{$I Dateiname}

6.7.2 Task-Auswahl

BESCHREIBUNG:	Mit diesem Compilerbefehl kann angegeben werden, in welcher Task (*TaskNr*, Werte 0..3) das entsprechende SAP-Programm ablaufen soll. Die Information wird im Autocodefile *„filename.cnc“* abgelegt. Mit Hilfe des PCAP-Befehls *txbf2()* wird dieses File automatisch in die richtige Task übertragen.
SYNTAX:	{$TASK *TaskNr*}
ANMERKUNG:	Sofern das entsprechende SAP-Programm diese Anweisung nicht enthält, wird die momentan selektierte Tasknummer zur Übersetzung herangezogen. Erfolgt aber der ($TASK)-Befehl, wird die entsprechend angewählte Tasknummer auch Default-Tasknummer für alle nachfolgenden Anzeige-, Start- und Stoppbefehle. Kapitel 3.2.1.

BEISPIEL:	*...*
	const
	Task1 = 1;
	...
	
	{$TASK Task1}; // oder
	{$TASK 1}

6.7.3 Full-System Compilierung

BESCHREIBUNG:	Mit diesem Compilerbefehl kann die Compiler-Option FULLSYSTEM angewählt werden.
SYNTAX:	{$FULLSYSTEM}
ANMERKUNG:	Sofern das entsprechende SAP-Programm diese Anweisung nicht enthält, wird die aktuell selektierte Option zur Übersetzung herangezogen. Diese Anweisung sollte am Anfang des Quelltextfiles stehen. Dieses Kommando ist ab mcfg V2.5.2.13 und ncc ab V2.5.2.9 verfügbar.

6.8 SAP-Laufzeitfehler

Bei der Ausführung von Stand-Alone-Programmen können verschiedene Fehler auftauchen. In diesem Fall wird die entsprechende Task angehalten. In der Datenstruktur CNCTS (siehe Kapitel 4.3.2.10) werden dann eine Fehlernummer und die Zeilennummer, in welcher der Fehler aufgetreten ist, eingetragen. Nachfolgend sind die Fehlernummern und mögliche Ursachen aufgelistet.

Tabelle 40: SAP-Laufzeitfehler

Fehler # dez / hex	Beschreibung
1 / 0001	Die arithmetische Operation ist ungültig für den verwendeten Datentyp
2 / 0002	Ungültiger Datentyp
4 / 0004	Ungültiger interner Operationscode. Dies kann auf ein Kompatibilitätsproblem zwischen mcfg/ncc und RWMOS.ELF zurückzuführen sein.
8 / 0008	Stack-Überlauf. Programm zu groß oder internes Problem der RWMOS-Betriebssystem-Software.
16 / 0010	Stack-Unterlauf. Dieser Fehler deutet auf ein internes Problem der RWMOS-Betriebssystem-Software hin.
32 / 0020	Unbekannter Event-Handler. Dies kann auf ein Kompatibilitätsproblem zwischen mcfg/ncc und RWMOS.ELF zurückzuführen sein.
64 / 0040	Ungültiges NC-Kommando. Dies kann auf ein Kompatibilitätsproblem zwischen mcfg/ncc und RWMOS.ELF zurückzuführen sein.
128 / 0080	Adressverletzung im NC-Programm. Dieser Fehler deutet auf ein internes Problem der RWMOS-Betriebssytem-Software hin.
256 / 0100	Adressverletzung im NC-Programm durch falsche Parameter-Angabe.
512 / 0200	Fehler bei Verwendung des AT-Interface
1024 / 0400	Eine Common-Variable außerhalb des gültigen Bereichs wurde adressiert.
2048 / 0800	Ungültiger Index bei einem Double-Zugriff auf den Common Buffer (nicht durch 8 teilbar).
4096 / 1000	Ungültiges SAP-Kommando. Dieser Fehler deutet auf ein Kompatibilitätsproblem zu Steuerungen einer anderen Generation hin.
8192 / 2000	Fehler bei Schnittgeschwindigkeitsinterpolation
16384 / 4000	Verwendung unerlaubter Achsen bei Interpolation mit G-Codes
32768 / 8000	Ungültiger Parameter bei arithmetischer Operation, z.B. mod 0
10000 hex	Zu viele SSF-Kommandos in einer Spline-Zeile (max. 8 sind möglich)
20000 hex	Rekursionstiefe bei G-Code-Unterprogrammen mit Programmwiederholung überschritten (O-Parameter)

x