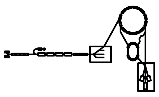

Einführung in NODAL

L. Hechler, U. Krause

GSI, Gesellschaft für Schwerionenforschung mbH
Planckstraße 1, D-64291 Darmstadt
Tel.: 06159/71-0

Issued: 25. Juli 2002
Key: U-NOD-03
Printed: 17. September 2014

Im Februar 1990 wurde in der GSI ein Kurs abgehalten, um den NODAL-Interpreter potentiellen Nutzern vorzustellen. Dabei sollten die wesentlichen Eigenschaften dargestellt und eine möglichst allgemeinverständliche Einführung in die Benutzung von NODAL gegeben werden. Die vorliegende Schrift entstand als Ausarbeitung dieses Kurses.

Entstehungsgeschichte		
Datum	Name	Kommentar
8. Jan. 1990	LH, UK	Beginn der Erstellung des Skripts
22. Feb. 1990	LH, UK	Vom Inhalt her abgeschlossen
1. Mär. 1990	LH, UK	Die letzten redaktionellen Feinheiten
16. Mär. 1990	LH, UK	Abgeschlossen
23. Jun. 1992	LH, UK	Zweite unveränderte Auflage
25. Jul. 2002	L. Hechler	Anpassungen für LaTeX2E and WWW

Inhaltsverzeichnis

1 Einführung

1.1 Wozu dient NODAL

1.1.1 Problemstellung

Teilchenbeschleuniger vom Umfang der GSI-Beschleunigeranlagen sind aus einigen hundert Geräten aufgebaut, die alle auf die korrekten Werte einzustellen und auf einen einwandfreien Betrieb zu überwachen sind. Um diese Aufgaben mit vernünftigem Aufwand bewältigen zu können, sind alle Geräte über ein Rechnernetz bedienbar.

Da die Einstellung und Überwachung in den meisten Fällen nach festen Regeln erfolgt, können diese Aufgaben gut von Bedienprogrammen übernommen werden. Dieser Weg wird in der GSI soweit möglich beschritten: für alle im Betrieb vorkommenden Aufgaben werden Programme erstellt, die dem Operateur Einzelheiten der Gerätebedienung weitgehend abnehmen und die Maschinensteuerung übersichtlich gestalten.

Es wird jedoch (insbesondere zu Beginn des Betriebes) vorkommen, dass noch nicht für alle Aufgaben ein entsprechendes Operatingprogramm zur Verfügung steht. Dann bleibt zunächst nur der Weg, zu dem entsprechenden Gerät hinzugehen und zu versuchen, es direkt einzustellen (soweit das überhaupt möglich ist) bzw. sich vor Ort über aufgetretene Fehler zu informieren. Das ist zumindest umständlich und wegen der weiten Wege auch sehr zeitaufwendig.

Weiter ist es sehr mühsam, für alle durchzuführenden Aktionen eigene Programme etwa in einer Sprache wie FORTRAN zu erstellen. Der Aufwand lohnt sich für die eigentlichen Operatingprogramme, da sie oft benötigt werden. Dagegen möchte man bei der Entwicklung neuer Geräte (und eventuell auch beim Ausprobieren neuer Steuerprozeduren) gerne alle Möglichkeiten über den Rechnerzugriff einmal durchtesten. Diese Testprozeduren werden aber nur kurze Zeit benötigt, zumindest wird man mehrere Möglichkeiten ausprobieren, bis man genau weiß, was erforderlich ist. Hier wäre der Weg über die Programmerstellung sehr aufwendig und stände in keinem Verhältnis zum eigentlichen Nutzen.

1.1.2 NODAL als flexibler Gerätezugriff

Für beliebige schnell durchführbare Gerätezugriffe steht in der GSI der NODAL-Interpreter zur Verfügung. Damit lässt sich auf jedes an das Kontrollsystem angeschlossene Gerät zugreifen. Alle Gerätezugriffe, die in den eigentlichen Operatingprogrammen möglich sind, können auch mit NODAL durchgeführt werden. Es lassen sich alle über das Kontrollsystem ansprechbaren Werte lesen (und auch verändern) sowie Funktionen wie ein Gerätereset durchführen.

NODAL ist als Interpreter konzipiert. Es ist damit vergleichbar zur Programmiersprache BASIC und sieht für den Benutzer auch ähnlich aus. Wie in BASIC werden in NODAL Befehle sofort ausgeführt, sodass man sich interaktiv über Geräte (und damit über die Beschleuniger) informieren und auch eingreifen kann (aber mit der nötigen Vorsicht!).

Ferner lassen sich auch kleine bis mittelgroße Programme in NODAL schreiben. Diese Programme lassen sich direkt ausführen. Bei der Abarbeitung der einzelnen Programmschritte werden diese analysiert und auf eventuelle Fehler wird sofort hingewiesen. Das Programm kann dann einfach abgeändert und sofort neu ausgeführt werden, da ein zeitraubendes separates Übersetzen und Binden nicht erforderlich ist. Zum Ändern der Programme sind verschiedene Möglichkeiten in NODAL schon vorgesehen (unter anderem zwei verschiedene Editoren), sodass der Interpreter nicht verlassen werden muss. Somit ist man sehr flexibel und kann sehr schnell Programme erstellen und austesten.

NODAL ist auf jedem an die Steuerrechner angeschlossenen Bildschirm lauffähig, benötigt also keine besondere Hardware wie Graphikbildschirme und kann daher von fast beliebigen Stellen innerhalb der GSI aufgerufen weden.

Als Nachteil ist zu nennen, dass die Programme relativ unübersichtlich werden, da NODAL nur eingeschränkte Möglichkeiten zur Strukturierung besitzt und somit insbesondere umfangreiche Programme schwer zu warten sind. Weiter sind die Möglichkeiten für Benutzerführungen in NODAL beschränkt, und die Programmausführung ist relativ langsam, da es sich um einen Interpreter handelt.

1.2 Ziel dieser Einführung

NODAL ist ein recht mächtiges Werkzeug. Entsprechend umfangreich ist die Anzahl der Befehle (weit über 100), von denen einige noch mehrere Varianten bieten.

Es wird hier nicht versucht, NODAL auch nur einigermaßen vollständig darzustellen. Beabsichtigt ist vielmehr, einen ersten Zugang zu geben und die Eigenschaften soweit zu erklären, dass man mit NODAL arbeiten kann. Ein Schwerpunkt wird dabei auf Gerätezugriffe im SIS Kontrollsystem gelegt.

NODAL ist in einigen Beschreibungen ausführlich dargelegt. Die vorliegende Schrift sollte so weit in NODAL einführen, dass diese Beschreibungen verständlich werden und man sich mit ihrer Hilfe weiter informieren kann. Die Beschreibungen, die es über NODAL gibt, sind im Anhang aufgeführt. Wenn man sich mit NODAL etwas vertraut gemacht hat, sollte man ruhig einmal in diesen Unterlagen blättern, um auch die Möglichkeiten von NODAL kennenzulernen, die in dieser Schrift nicht erwähnt werden konnten.

Ein Teil dieser Ausarbeitung wird sich mit der Erstellung von NODAL-Programmen befassen, was wahrscheinlich nicht für jeden von Interesse sein wird. Aber die meisten Nutzer von NODAL werden sicher irgendwann bereits fertige Programme aufrufen und abarbeiten lassen. Deshalb sei empfohlen, sich zumindest mit den Grundlagen und Prinzipien der Programmierung zu beschäftigen, um besser zu verstehen, was beim Aufruf eines NODAL-Programms abläuft.

1.2.1 Vorgehensweise

Die Befehle und Eigenschaften werden nicht streng systematisch erläutert. Es wird in der Regel jeweils nur das dargestellt, was aktuell benötigt wird. Dafür wird oft schon Bekanntes erneut aufgegriffen und vertieft.

Soweit möglich, werden die Darstellungen durch Beispiele erläutert. Dabei ist angestrebt, die Beispiele so wiederzugeben, wie sie auf dem Terminalbildschirm aussehen. Zur besseren Unterscheidung sind die Teile, die der Benutzer eingibt, kursiv wiedergegeben (wie *`Eingabe`*), die Ausgaben vom Rechner werden in normaler Schrift wiedergegeben (wie `Ausgabe`).

Generell werden alle Befehle durch Drücken der Taste `RETURN` abgeschlossen und damit ausgelöst. Im Bildschirmbild bewirkt sie, dass eine neue Zeile begonnen wird. Die `RETURN`-Taste wird in den aufgeführten Beispielen nicht explizit angegeben.

2 Erste Kontakte mit NODAL

2.1 Starten von NODAL

NODAL kann von jedem an einen der Steuerrechner angeschlossenen VAX-Terminal gestartet werden. Daneben besteht auch die Möglichkeit, NODAL von den Touchpannels der Beschleunigerkonsolen zu starten.

2.1.1 Start per Touchpannel

Das ist die einfachste Art, NODAL zu starten. Es muss lediglich das Feld mit der Aufschrift NODAL gedrückt werden. Da die Eingaben für NODAL über die Tastatur erfolgen, ist natürlich darauf zu achten, dass dem Bildschirm eine Tastatur zugeordnet ist.

2.1.2 Start vom Terminal

Um NODAL von einem normalen Terminal zu starten, muss man sich zuvor beim Rechner angemeldet haben (auf neudeutsch: man muss „eingeloggt“ sein). Damit NODAL arbeitet, müssen weiter einige logische Namen definiert sein. Da eine ausreichende Darstellung der erforderlichen Einstellungen den Rahmen dieser Schrift sprengen würde, wird bei auftretenden Schwierigkeiten um Rücksprache bei der Prozessrechnergruppe gebeten.

Eine Einstellung, unter der NODAL lauffähig ist, erhält man automatisch, wenn man sich unter dem Namen `SISUSER` anmeldet, was im folgenden erläutert werden soll.

Als erstes ist bei einem unbenutzten Terminal die Taste `RETURN` einmal zu drücken. In der Regel erscheint dann die Kennung des „Servers“, des Gerätes, das die Verbindung zum Rechner aufbaut, zusammen mit der Meldung „`Local`“ (bei einigen Terminals erscheint allerdings sofort eine Meldung wie „`VAX 8700 ALICE : Please login`“ - in diesem Fall ist direkt weiterzumachen wie unten beschrieben). Die Meldung „`Local`“ bedeutet, dass man sich erst einmal durch Eingabe des Buchstabens „`C`“ (für „connect“) und anschließendem Drücken der Taste „`RETURN`“ zu einem Rechner durchschalten muss. Dabei sind die Terminals (als Voreinstellung, die man natürlich umändern kann) unterschiedlichen Rechnern zugeordnet. Im Bereich der Gerätesteuerung ist das der Rechner *ALICE* oder die Rechner des Hauptkontrollraum-Clusters, auf denen jeweils NODAL benutzt werden kann.

Nach dem Durchschalten zu einem Rechner erscheint die Meldung dieses Rechners mit der Aufforderung, den Benutzernamen („Username“) und danach das Kennwort („Password“) einzugeben. Für beides ist `SISUSER` einzutippen, wobei das Kennwort nicht auf dem Bildschirm erscheint, und jeweils mit der `RETURN`-Taste abzuschliessen. Es ergibt sich dabei folgendes auf dem Bildschirm, wobei als Beispiel der Rechner *ALICE* gewählt wurde:

```
S200BO Local> c
Local -010- Session 1 to V780A established


VAX 8700 ALICE : Please login

Username: SISUSER
Password: SISUSER
Welcome to VAX/VMS ...
```

Nun erscheint eine Reihe von Meldungen und bis zum Ende der Anmeldeprozedur dauert es etwas.

Ist sie abgeschlossen, erscheint die Rechnerkennung und die Eingabeaufforderung („prompt“) der VAX, das Dollarzeichen:

```
A $
```

Das „A“ zeigt an, dass man auf dem Rechner *ALICE* arbeitet. Die entsprechende Kennung für die HKR-Rechner ist der Buchstabe H, gefolgt von der Nummer des Rechners. So erscheint auf dem Rechner *HKR22* die Meldung

```
H22 $
```

Nach dieser Eingabeaufforderung ist NODAL einzugeben (und mit der RETURN-Taste abzuschließen), um NODAL zu starten.

```
A $ NODAL
```

Bei allen hier erläuterten Eingaben können beliebig Klein- oder Großbuchstaben verwendet werden.

2.2 Erscheinungsbild

Nach dem Starten erscheint die Versionskennung von NODAL und danach die NODAL-Eingabeaufforderung („prompt“), der Winkel >:

```
NODAL in MODULA-2, GSI Version 1.05 - 11. Jan. 90

>
```

Immer, wenn dieser Winkel erscheint, können Befehle eingegeben werden. Sie werden ausgeführt, nachdem sie durch Drücken der RETURN-Taste abgeschlossen werden.

Bis dahin können sie noch verändert und korrigiert werden: Mit den vertikalen Richtungs-Tasten (Cursor-Tasten) können die letzten 20 Eingabezeilen zurückgeholt werden, mit den beiden horizontalen Richtungs-Tasten und der Löschtaste (DELETE, die beschriftet ist etwa wie <X|) kann die aktuelle Eingabe verändert werden. Das ist recht hilfreich, wenn kurz hintereinander ähnlich lautende Befehle einzugeben sind.

2.3 Beenden

Wird NODAL aufgerufen, sollte es auch wieder verlassen werden können. Dazu dient der Befehl QUIT. Nachdem er eingegeben wurde, werden einige statistische Informationen ausgegeben und NODAL beendet. Als Zeichen, dass man sich wieder auf der VAX-Ebene befindet, erscheint die Rechnerkennung (hier der Buchstabe A) und das Dollarzeichen:

```
>QUIT

ELAPSED: 0 00:00:27.12 CPU: 0:00:01.27 BUFIO: 38 DIRIO: 12 FAULTS: 380

A $
```

Es soll schon an dieser Stelle erwähnt werden, dass der Befehl QUIT auch bis auf den Buchstaben Q abgekürzt werden kann.

Nachdem alle Arbeiten beendet sind, sollte man sich auch wieder vom Rechner abmelden (wenn man sich eigens angemeldet hat, also NODAL nicht per Touchpannel gestartet wurde). Dazu dient der VAX-Befehl LO:

```
A $ LO
```

der in einer Meldung resultiert wie

```
  SISUSER       logged out at 21-Jan-1990 15:01:37.75
Local  -011- Session 1 disconnected from V780A
S00BO Local>
```

2.4 Unterbrechen

Wie schon erwähnt, können NODAL-Befehle nur eingegeben werden, wenn die NODAL-Eingabeaufforderung (der Winkel `>`) am Anfang der Eingabezeile erscheint.

Manchmal soll aber ein Programm (oder auch ein NODAL-Befehl) vorzeitig abgebrochen werden, etwa weil versehentlich ein falsches gestartet wurde oder das Programm (etwa wegen eines Fehlers) für alle Zeiten weiterlaufen würde. Dazu können keine „normalen“ NODAL-Befehle verwendet werden, da zu deren Abarbeitung das Programm erst beendet sein muss.

Um dennoch zur Eingabeanforderung von NODAL zurückkehren zu können, ist die Flucht-Taste vorgesehen. Es ist die meist als `ESC` (für „escape“) beschriftete Taste `F11` in der obersten Tastenreihe (über der Text-Tastatur). Durch Drücken dieser Taste wird die Ausführung eines NODAL-Programms und der meisten Befehle abgebrochen und NODAL ist wieder bereit, Befehle entgegenzunehmen, was durch den Eingabewinkel angezeigt wird. Dabei erscheint die Meldung

```
***  NODAL ERROR  16   Escape Typed
     >
```

Als Hinweis für diejenigen, die schon auf einer VAX gearbeitet haben, sei erwähnt, dass in NODAL die von der VAX gewohnten Unterbrechungen mit den Tastenkombinationen `Ctrl-Y` und `Ctrl-C` nicht funktionieren und diese Tasten teilweise sogar eine andere Bedeutung haben.

2.5 Fehlermeldungen

Natürlich kann man mal etwas falsch machen, etwa wenn man einen Befehl falsch eingibt oder versucht, durch Null zu dividieren. Immer, wenn NODAL eine Unkorrektheit feststellt, erscheint die Meldung `NODAL ERROR` mit einer Nummer und etwas Text, der den Fehler charakterisiert und erläutert.

Gibt man z.B. statt `QUIT` versehentlich `KWIT` ein, so stellt NODAL fest, dass es keinen Befehl `KWIT` gibt und zeigt es an durch

```
> KWIT

***  NODAL ERROR  8    Nonexistent Name
>
```

Nach der Ausgabe des Fehlers ist NODAL wieder bereit, Befehle entgegenzunehmen, was durch die Eingabeaufforderung, den Winkel (`>`), angezeigt wird.

Ein Fehler, der häufig auftreten wird und vielleicht etwas Ratlosigkeit hinterlässt, ist der Fehler Nummer 41 (`"Syntax Error"`). Er bedeutet, dass NODAL die Eingabe nicht interpretieren kann, weil sie nicht der erforderlichen Struktur eines Befehles entspricht. Der Grund kann etwa sein, dass ein Komma vergessen wurde oder in einem Befehl ein Zeichen auftritt, das dort nicht vorgesehen ist.

Beispielsweise tritt der Fehler auf, wenn nach dem Befehl `QUIT` versehentlich ein weiteres Zeichen eingegeben wurde:

```
>QUIT :

***  NODAL ERROR 41    Syntax Error
>
```

NODAL besitzt eine nützliche Hilfe, um anzuzeigen, wo der Fehler *festgestellt* wurde (das ist meistens, allerdings nicht immer, die Stelle, die falsch ist). Dazu sind, direkt nachdem der Fehler aufgetreten ist, *gleichzeitig* die Tasten `CTRL` (Steuerung, „control“) und `B` zu drücken (erst `CTRL` drücken und festhalten, dann `B` drücken). Dann wird die fehlerhafte Zeile angezeigt, wobei die Schreibmarke („cursor“) an der Stelle steht, an der der Fehler festgestellt wurde. Im obigen Beispiel (`QUIT` mit Doppelpunkt) würde nach dem Drücken von `Ctrl-B` die Zeile erneut angezeigt, wobei die Schreibmarke auf den Doppelpunkt zeigt.

3 Erste Befehle

3.1 Einführung

Der Befehl `QUIT` und die Flucht-Taste (`ESC`) sind die Hilfsmittel, mit denen man sich bei auftretenden Problemen notfalls helfen kann und zumindest in der Lage ist, seinen Arbeitsplatz wieder geordnet zu verlassen.

Nun aber sollen endlich einige „richtige“ Befehle vorgestellt werden, um wirklich mit NODAL arbeiten zu können.

Dabei sind zwei unterschiedliche Arbeitsmodi von NODAL zu unterscheiden, die als Direkt-Modus und als Programm-Modus bezeichnet werden.

Direkt-Modus:
: Jeder Befehl wird für sich eingegeben und sofort abgearbeitet.

Programm-Modus:
: Mehrere Befehle werden zu einem Programm zusammengefasst; diese werden nach dem Starten des Programms nacheinander abgearbeitet, ohne dass weitere Befehle eingegeben werden müssen.

Zunächst wird der Direkt-Modus erläutert, da er einfacher zu handhaben und zu erklären ist.

3.2 Befehlsformat

Wie schon erwähnt, sind Befehle nach der NODAL-Eingabeaufforderung, dem Winkel, einzugegeben. Ein Befehl besteht aus einem Befehlswort und in der Regel weiteren Angaben.

Befehle können beliebig in Groß- und Kleinbuchstaben geschrieben werden, dazwischen wird nicht unterschieden. Viele Befehlswörter können soweit abgekürzt werden, wie sie noch eindeutig sind. Wo das möglich ist, wird darauf hingewiesen.

Leerzeichen dürfen nicht innerhalb von Namen auftreten, können aber zu Beginn einer Zeile und zwischen den einzelnen Elementen der Befehle in der Regel beliebig eingefügt werden.

Alle Befehle werden durch Drücken der Taste `RETURN` abgeschlossen.

Aus methodischen Gründen werden alle Befehle und reservierten Namen in den hier angegebenen Beispielen groß geschrieben und die Befehle nicht abgekürzt.

Als Erinnerung: In den Beispielen werden Benutzereingaben *`geneigt`* wiedergegeben, Ausgaben vom Rechner `gerade`. Der Abschluss der Befehle durch die `RETURN`-Taste ist dabei in der Regel nicht explizit angegeben.

3.3 Befehl `TYPE`

Dieser Befehl dient zur Ausgabe auf dem Bildschirm.

Hinter dem Befehlswort `TYPE` sind die auszugebenden Ausdrücke anzugeben, die jeweils durch (ein oder mehrere) Leerzeichen voneinander getrennt werden müssen. Das Befehlswort kann bis auf `T` abgekürzt werden.

Ausgegeben werden können z.B. Zahlen:

```
>TYPE 17
    17.0000
>TYPE 17 4
    17.0000     4.0000
>
```

was natürlich nicht sehr interessant ist.

Statt Zahlen lassen sich aber auch beliebige mathematische Ausdrücke verwenden, deren Ergebnis ein numerischer Wert ist. Diese Ausdrücke werden berechnet und das Ergebnis ausgegeben. Als Operatoren sind Addition (`+`), Subtraktion (`-`), Multiplikation (`*`), Division (`/`) und Potenzierung (`^`) vorgesehen.

Gerechnet wird in der üblichen Reihenfolge. Erst werden Potenzierungen durchgeführt, dann gleichwertig Multiplikationen und Divisionen und zum Schluss Additionen und Subtraktionen. Sind Ausdrücke durch gleichwertige Operatoren verknüpft, werden die Ausdrücke von links nach rechts abgearbeitet. So wird in dem Ausdruck 2/4*3 erst 2 durch 4 dividiert und das Resultat 0.5 anschließend mit 3 multipliziert, so dass sich als Gesamtergebnis 1.5 ergibt. Soll von der impliziten Ordnung abgewichen werden, wird durch die Angabe von Klammern (`(...)`) bewirkt, dass die Ausdrücke innerhalb der Klammern zuerst abgearbeitet werden.

```
>TYPE 17+4
    21.0000
>TYPE  2^3
     8.0000
>TYPE  5  3*(1+1)  3*1+1
     5.0000     6.0000     4.0000
>
```

Wie die Ausgabe der Ergebnisse schon vermuten lässt, können auch Gleitkommazahlen angegeben werden. Dabei ist der Nachkommateil durch einen Punkt abzutrennen (ein Komma wird als Trennzeichen zwischen zwei Ausdrücken interpretiert). Für große Zahlen ist eine exponentielle Darstellung günstiger. Dabei wird das übliche Format verwendet, bei dem der Exponent (zur Basis 10) durch den Buchstaben `E` (oder `e`, da NODAL nicht zwischen Klein- und Großbuchstaben unterscheidet) gekennzeichnet wird. So lässt sich die Zahl $5.2 \cdot 10^3$ in NODAL darstellen als `5.2E3` und die Zahl 47.11 lässt sich etwa angeben durch:

```
>TYPE 47.11  4711E-2  4.711E1  471.1e-1
    47.1100    47.1100    47.1100    47.1100
>
```

Eine entsprechende Darstellung im Exponentialformat wird bei der Ausgabe automatisch gewählt, wenn die Zahl so groß ist, dass sie bei der Ausgabe als Fließkommazahl sehr lang werden würde:

```
>TYPE 4711*8150
3.8395E7
>
```

Der genaue Wert von 38394650 würde zur Darstellung als Fließkommazahl mehr Platz benötigen, als NODAL standardmäßig vorsieht, daher wird bei der Ausgabe auf die exponentielle Darstelllung umgeschaltet. Natürlich kann man das Ausgabeformat und auch die Darstellung (fast) beliebig verändern, aber um diesen Abschnitt nicht zu umfangreich werden zu lassen, sei diesbezüglich auf das Kapitel über Formatierungen verwiesen.

Erwähnt werden soll aber noch, dass NODAL die Zahlen bei der Ausgabe auf die ausgegebene Länge rundet, wie man an dem letzten Beispiel sehen kann.

NODAL rechnet intern mit einer Gleitkommadarstellung mit 55 Bit für die Mantisse und 8 Bit für den Exponenten. Das entspricht einem Zahlenbereich von etwa $-1.7 \cdot 10^{38}$ bis $+1.7 \cdot 10^{38}$

mit einer Genauigkeit von etwa 16 Stellen. Das erscheint übertrieben, erlaubt aber, auch 32 Bit Ganzzahlwerte (die maximal 10-stellig sein können) ohne Rundungsfehler im niederwertigsten Bit zu verarbeiten.

Als Hinweis sei erwähnt, dass auch eine Reihe mathematischer Funktionen wie die Quadratwurzel `SQR` und der Sinus `SIN` zur Verfügung stehen, die wie Zahlen in Ausdrücken verwendet werden können. Sie sind durch den Namen anzugeben, wobei das Argument hinter dem Namen in Klammern folgen muss. Zwischen dem Namen der Funktion und der zugehörigen Klammer darf kein Leerzeichen sein:

```
>TYPE (1/2)*SQR(2)
     .7071
>TYPE SIN(45*(3.14159/180))
     .7071
>
```

Die wichtige Kreiszahl π steht unter dem Namen `PIE` ebenfalls zur Verfügung:

```
>TYPE  PIE  SIN(45*(PIE/180))
     3.1416      .7071
>
```

Die Liste der in NODAL vorhandenen Funtionen findet man z.B. in [2].

3.4 Befehl `SET`

Dieser Befehl dient dazu, Variablen numerische Werte zuzuweisen. Er kann abgekürzt werden bis auf `SE`.

Das beinhaltet, dass in NODAL Variable verwendet werden können. Eine Variable ist ein Speicherplatz, der einen Namen bekommt und in dem man einen (in diesem Fall numerischen Wert) ablegen kann. Auf diesen Wert kann man über den Namen der Variablen zugreifen. Man kann ihr einen Wert zuweisen, also den Wert in der Variablen ablegen; diesen Wert verändern, indem man der Variablen einen neuen Wert zuweist; und diesen Wert auch in numerischen Ausdrücken weiter verwenden. Das geschieht, indem man den Namen der Variablen wie eine Zahl in einem numerischen Ausdruck benutzt. Dabei wird der Wert, der gerade unter diesem Namen abgespeichert ist, anstelle des Variablennamens in den Ausdruck eingesetzt und bei der Berechnung des Ergebnisses benutzt.

Um einer Variablen einen Wert zuzuweisen, ist hinter dem Befehlswort `SET` der Name der Variablen und danach hinter einem Gleichheitszeichen der (neue) Wert der Variablen anzugeben. Dieser Wert kann direkt eine Zahl sein, kann aber auch durch einen mathematischen Ausdruck gegeben sein, der seinerseits wiederum Variablen enthalten kann.

So werden durch

```
>SET Wert=17
>SET Vier=2+2
>
```

den beiden Variablen `Wert` und `Vier` neue Werte zugewiesen, die man sich z.B. mit dem Befehl `TYPE` anschließend ansehen kann:

```
>TYPE Wert
    17.0000
>TYPE Vier
     4.0000
>
```

Gibt es eine Variable noch nicht, wird sie angelegt, also der Speicherplatz eingerichtet, wenn ihr das erste Mal mit dem Befehl `SET` ein Wert zugewiesen wird. Das bedeutet insbesondere, dass einer Variablen einmal ein Wert zugewiesen sein muss, bevor sie in einem Ausdruck verwendet werden kann:

```
>SET zwei=2
>TYPE 2*zwei
     4.0000
>
```

Namen für Variablen dürfen aus einem bis acht Zeichen bestehen, wobei Buchstaben (`A..Z, a..z`), Ziffern (`0..9`), Punkt (`.`) und Unterstrich (`_`) erlaubt sind. Das erste Zeichen eines Variablennamens muss allerdings ein Buchstabe sein. Erlaubt sind also z.B. `Strom`, `BIT12`, `Time_3`, nicht dagegen `_xyz` sowie `17und4`.

Groß- und Kleinschreibung kann beliebig verwendet werden, sie werden nicht unterschieden. So bezeichnet `VarName` und `VARNAME` die gleiche Variable. Allerdings sei empfohlen, durch eine gemischte Klein- und Großschreibung das Schriftbild sinnvoll zu gliedern, um die Lesbarkeit zu erhöhen: so lässt sich `StromNeu` leichter lesen als `STROMNEU` oder gar `sTrOmnEu`, obwohl alle drei die gleiche Variable bezeichnen.

Vordefinierte Funktionen wie `SIN`, `ABS` usw. sowie Namen (Nomenklaturen) von Geräten im Beschleunigerkontrollsystem können nicht als Variablennamen benutzt werden.

3.5 Befehl `DIMENS`

Der Befehl `TYPE` und auch der Befehl `SET` sind bereits im Direkt-Modus sehr nützlich. Mit ihnen kann man NODAL schon wie einen recht komfortablen Taschenrechner benutzen. Der Befehl `DIMENS` wird dagegen im Direkt-Modus nicht häufig verwendet. Da er aber für viele Gerätezugriffe erforderlich ist, wird er schon hier erläutert.

Er dient zum Vereinbaren und Anlegen eines Feldes („array“). Er kann abgekürzt werden bis auf `DI`.

Die bisher behandelten Variablen dienten dazu, jeweils einen Wert zu speichern. Im Gegensatz dazu lassen sich in einem Feld, das wie die einfachen Variablen über einen Namen angesprochen werden kann, mehrere Werte abspeichern, die durchnummeriert werden und auf die einzeln zugegriffen werden kann.

Soll ein Feld angelegt werden, müssen sein Name sowie die Anzahl der Elemente, also die Anzahl der Werte, die es enthalten kann, vereinbart werden. Das geschieht, indem hinter dem Befehlswort `DIMENS` der Feldname und dahinter in runden Klammern die Zahl der Elemente angegeben werden.

Formal lässt sich dieser Befehl beschreiben durch

```
DIMENS <Feldname> ( <Feldgröße> )
```

So wird mit

```
>DIMENS Vektor(6)
>
```

ein Feld mit dem Namen `Vektor` vereinbart, das 6 Elemente enthält.

Auf diese Elemente lässt sich zugreifen, indem hinter den Feldnamen in runden Klammern die Nummer des Elements (der Feldindex[1]) gesetzt wird. Diese so bezeichneten Elemente können wie Variablen benutzt werden. Mit dem soeben definierten Feld lässt sich also schreiben:

[1] Feldindizes haben in NODAL stets 1 als untere Grenze

```
>SET Vektor(3)=2
>SET Vektor(5)=Vektor(3)+3
>TYPE Vektor(5)
     5.0000
>TYPE Vektor(5)/Vektor(3)
     2.5000
>
```

Zum Schluss sei noch der Hinweis gegeben, dass auch zweidimensionale Felder (also Matrizen) möglich sind. Hier sind bei der Dimensionierung zwei Feldgrenzen und beim Zugriff zwei Indizes (beidemale durch Komma getrennt) anzugeben:

```
>DIMENS Matrix(2,3)
>SET Matrix(1,2)=2
>SET Matrix(1,3)=Matrix(1,2)*3
>TYPE Matrix(1,3)
     6.0000
>
```

3.6 Befehl `FOR`

Bei Feldern mit vielen Elementen ist es sehr mühsam, sich deren Inhalt anzusehen, indem man jedes Element in einem eigenen `TYPE`-Befehl ausgibt. Es wäre daher wünschenswert, mit einem Befehl den gesamten Inhalt auszugeben.

Das ist zum Beispiel mit einer Laufanweisung möglich, die in NODAL durch den Befehl `FOR` gegeben ist.

Mit ihm kann ein NODAL-Befehl mehrfach ausgeführt werden. Um erkennen zu können, wie oft der Befehl schon ausgeführt wurde, wird dabei die Anzahl der Durchläufe in einer Variablen mitgezählt, der Laufvariablen. Diese Laufvariable ist eine ganz gewöhnliche NODAL-Variable, die dementsprechend auch innerhalb des Schleifenbefehls benutzt werden kann. Sie wird zu Beginn der Ausführung des Befehls `FOR` auf einen anzugebenden Anfangswert gesetzt. Nach jedem Durchlauf wird sie um eins erhöht und anschließend die Schleife erneut durchlaufen, bis die Laufvariable den anzugebenden Endwert erreicht hat.

Im Befehl `FOR` ist nach dem Befehlswort `FOR` der Name der Laufvariable anzugeben, dahinter nach einem Gleichheitszeichen durch ein Komma getrennt der Anfangswert und der Endwert der Laufvariablen. Anschließend folgt, durch ein Semikolon getrennt, der Befehl, der bei jedem Durchlauf ausgeführt werden soll. In diesem Befehl kann die Schleifenvariable, wie schon gesagt, wie eine normale Variable verwendet werden.

In einer formalen Beschreibung ist dieser Befehl gegeben durch

```
FOR <Laufvariable> = <Anfangswert> , <Endwert> ; <NODAL-Befehl>
```

Er kann abgekürzt werden bis auf `F`.

So lässt sich das in der vorigen Abschnitt angelegte und teilweise mit Werten versehene Feld `Vektor` ausgeben durch:

```
>FOR LaufVar=1,6;TYPE Vektor(LaufVar)
     0.0000     0.0000     2.0000     0.0000     5.0000     0.0000
>
```

Bei der Abarbeitung dieses Befehls erhält die hier `LaufVar` genannte Laufvariable nacheinander die Werte 1, 2, ..., 6. Jedesmal, also insgesamt 6 mal, wird mit diesem Wert der Laufvariablen der Befehl `TYPE Vektor(LaufVar)` ausgeführt, so dass nacheinander alle Elemente des Feldes

ausgegeben werden, also `Vektor(1)`, `Vektor(2)`, ..., `Vektor(6)`.

Natürlich kann nicht nur der Befehl `TYPE` in einer Laufanweisung verwendet werden, sondern jeder NODAL-Befehl wie z.B. der Befehl `SET`, mit dem im folgenden Beispiel den Elementen des neu anzulegenden Feldes `Ganz` die ganzen Zahlen zugewiesen werden:

```
>DIMENS Ganz(5)
>FOR Index=1,5;SET Ganz(Index)=Index
>
```

die angezeigt werden durch:

```
>FOR Index=1,5;TYPE Ganz(Index)
     1.0000     2.0000     3.0000     4.0000     5.0000
>
```

Die Laufvariable muss nicht (darf aber) zuvor mit einer Wertzuweisung über `SET` angelegt worden sein. Gibt es sie noch nicht, wird sie neu angelegt.

Als Hinweis sei noch erwähnt, dass in der Laufanweisung nicht nur *ein* Befehl abgearbeitet werden kann, sondern mehrere. Wie das im einzelnen aussieht, wird noch behandelt werden. Für ganz Ungeduldige: Eine Möglichkeit besteht darin, statt eines Befehles mehrere durch Semikola getrennt anzugeben.

3.7 Befehl `TYPE` – Ergänzungen

An den letzten Beispielen wurde ersichtlich, dass der Befehl `TYPE` in einer Laufanweisung nicht automatisch jedesmal in eine neue Zeile schreibt. Alle Ausgaben erfolgen daher in der gleichen Bildschirmzeile nebeneinander. Bei vielen Ausgaben, etwa wenn ein großes Feld angezeigt werden soll, würde dabei versucht, über den rechten Bildschirmrand hinauszuschreiben. Das hätte zur Folge, dass die letzten Ausgaben nicht mehr sichtbar sind, da am Bildschirmrand abgeschnitten wird.

In NODAL gibt es aber eine Möglichkeit, bei dem `TYPE`-Befehl anzugeben, dass eine neue Zeile begonnen weden soll. Dazu ist in die Liste der auszugebenden Elemente ein Ausrufungszeichen (`!`) einzufügen. Es bewirkt, dass an dieser Stelle mit einer neuen Zeile begonnen wird:

```
>TYPE 5  7
     5.0000     7.0000
>
```

aber

```
>TYPE 5 ! 7
     5.0000
     7.0000
>
```

Das Ausrufungszeichen ist eine der möglichen Formatierungen in NODAL. Mit ihm kann man nun längere Ausgaben bei Laufanweisungen auf dem Bildschirm darstellen.

Als Beispiel sollen in einem neu anzulegendem Feld `Quadrat` mit einer Laufanweisung die Quadrate der Zahlen von 1 bis 10 abgespeichert werden

```
>DIMENS Quadrat(10)
>FOR Index=1,10;SET Quadrat(Index)=Index*Index
>
```

und anschließend in einer Laufanweisung zusammen mit der Laufvariablen angezeigt werden, wobei die Formatierung durch das Ausrufungszeichen bewirkt, dass für jede Quadratzahl eine neue Zeile

begonnen wird:

```
>FOR Index=1,10;TYPE Index Quadrat(Index) !
     1.0000     1.0000
     2.0000     4.0000
     3.0000     9.0000
     4.0000    16.0000
     5.0000    25.0000
     6.0000    36.0000
     7.0000    49.0000
     8.0000    64.0000
     9.0000    81.0000
    10.0000   100.0000
>
```

Den Betrachter werden dabei vielleicht die vielen Nullen stören. Eigentlich sind nur natürliche Zahlen angegeben worden, aber NODAL fasst sie als reelle Zahlen auf und gibt sie entsprechend mit vier Stellen nach dem Komma aus. Leider kann NODAL nur reelle Zahlen verarbeiten, es ist nicht vorgesehen, mit ganzen Zahlen zu rechnen. Aber es besteht immerhin die Möglichkeit, Werte als ganze Zahlen auszugeben. Dazu ist vor einem auszugebenden Wert (das kann explizit eine Zahl sein oder auch ein numerischer Ausdruck) eine Formatierung der Form `%n` anzugeben. Sie bewirkt, dass der Wert als ganze Zahl mit `n` Ziffern ausgegeben wird. Bei der Ausgabe eines Wertes wird dieser gerundet, ist der Wert kleiner als die angegebene Zahl von Ziffern, werden Leerzeichen aufgefüllt.

```
>TYPE %5 17 %3 3.6
   17  4
>
```

Mit dieser Formatierung lässt sich das obige Beispiel etwas schöner gestalten, indem die Werte als ganze Zahl mit jeweils maximal 4 Ziffern ausgegeben werden:

```
>FOR Index=1,10;TYPE %4 Index Quadrat(Index) !
   1   1
   2   4
   3   9
   4  16
   5  25
   6  36
   7  49
   8  64
   9  81
  10 100
>
```

Wie man an diesem Beispiel sieht, gilt diese Formatierung innerhalb eines `TYPE`-Befehls so lange, bis sie durch eine neue geändert wird.

Während die genannte Formatierung zwar die Ausgabe etwas schöner gestaltet, aber keine neuen Informationen liefert, sollen nun weitere Formatierungen erwähnt werden, die oft zur Interpretation der Daten sehr nützlich sind.

Insbesondere bei der Gerätesteuerung hat man es teilweise mit Werten zu tun, die bitweise zu interpretieren sind, bei denen also zum Beispiel jedes Bit eines Statuswortes eine eigene Bedeutung hat. Um solche Bitmuster einfacher interpretieren zu können, gibt es die Möglichkeit, Werte hexadezimal oder bitweise auszugeben.

Um das Ergebnis eines Ausdrucks hexadezimal auf dem Bildschirm auszugeben, sind ihm zwei

schließende eckige Klammern (]]) voranzustellen. Entsprechend wird durch ein vorangestelltes Fragezeichen (?) das Ergebnis eines Ausdrucks bitweise dargestellt. In beiden Fällen wird der Wert zuvor auf ganze Zahlen gerundet.

```
>TYPE ]]27
0000001B
>TYPE ?27
00000000000000000000000000011011
>
```

Auch das Umgekehrte ist teilweise möglich. Man kann in Ausdrücken, insbesondere also im Befehl TYPE, Zahlen auch hexadezimal angeben (binär ist nicht möglich). Dazu sind einer Zahl zwei öffnende eckige Klammern ([[) voranzustellen, damit sie als Hexadezimalzahl interpretiert wird.

Als Beispiel sei die hexadezimale Zahl $7F_{hex}$ genannt, die dezimal als 127 dargestellt ist. Sie wird einmal direkt ausgegeben, zum anderen in einer Rechnung benutzt:

```
>TYPE [[7F
   127.0000
>TYPE 2*[[7F
   254.0000
>
```

Mit den Möglichkeiten, numerische Werte auch hexadezimal sowohl ein- als auch auszugeben, kann man viele Umrechnungen, die insbesondere bei gerätenahen Anwendungen auftreten, vereinfachen.

Als letzter Hinweis in diesem Abschnitt soll erwähnt werden, das mit dem Befehl TYPE nicht nur Zahlen, sondern auch Texte ausgegeben werden können. Sie sind dazu in Hochkommas (") einzuschließen:

```
> TYPE 17 -und-4
    17.0000 und     4.0000
>
```

4 Gerätebedienung

Gerätebedienung ist ein sehr weit gefasster Begriff. Die Anforderungen, die an eine Gerätebedienung gestellt werden, sind sehr unterschiedlich, abhängig von dem, der sie stellt.

Technische Anforderungen werden von den Geräten selbst gestellt. Bei der Einschaltprozedur eines Netzgerätes muss ein Zeitverhalten berücksichtigt werden; nur eine bestimmte Steuersequenz setzt ein Gerät zurück (Reset); ADCs und DACs haben eine Skalierung und erwarten bzw. liefern entsprechende Bitmuster; und vieles vieles mehr.

Ebenfalls technische Anforderungen stellen die verschiedenen Rechner. Bei der Kommunikation der Rechner müssen bestimmte Protokolle eingehalten werden; Adressen müssen bekannt sein, um Pakete an ihr Ziel zu bringen; Überwachungsaufgaben sind gefordert; und auch hier noch vieles vieles mehr.

Völlig andere Anforderungen stellen die Benutzer der Geräte und Rechner. Für sie sind die für die Beschleunigerphysik wichtigen Eigenschaften der Geräte (Hochspannung, Flankensteilheit, Flattoplänge, Chopperfenster, Sondenstrom, ...) von Interesse. Und natürlich möchten sie in der Regel auf diese Eigenschaften auch in „vernünftigen“ physikalischen Einheiten zugreifen (also Volt, Ampère, Sekunden usw.). Das Wissen um DAC-Bitmuster, IFK-Funktionskodes, Steuersequenzen für I/O-Bausteine, Kommunikationsprotokolle und ähnliches ist hier dagegen überhaupt nicht gefragt.

Diese unterschiedlichen Anforderungen bringt das Kontrollsystem unter einen Hut. Es übersetzt Gerätenamen in Adressen, Eigenschaften in Verarbeitungsvorschriften, ADC-Bitmuster in Stromwerte, Kommandos in Kommunikationsprotokolle und so einiges mehr. Es fungiert als Schnittstelle zwischen Benutzern auf der einen und Geräten auf der anderen Seite und wählt, je nachdem mit welcher Seite es gerade spricht, die passende Sprache.

Damit erleichtert das Kontrollsystem den Zugriff auf Geräte und gestattet dem Benutzer die Konzentration auf deren wesentliche Eigenschaften.

4.1 Gerätezugriffe mit NODAL

Die Besonderheit von NODAL gegenüber kommerziellen Interpretern wie etwa BASIC liegt in der Möglichkeit, direkt Gerätezugriffe durchführen zu können. Dabei werden die Funktionen der Kontrollsystem-Schnittstelle noch einmal wesentlich vereinfacht.

Was im einzelnen nötig ist, um einen Gerätezugiff in NODAL zu formulieren, soll im Folgenden aufgeführt werden.

Nomenklatur: Jedes Gerät zur Steuerung der Beschleuniger bei GSI hat einen eindeutigen Namen, eine *Nomenklatur*, die nach bestimmten Prinzipien aufgebaut ist (siehe GSI-Nomenklatursystem von Schaa und Peldzinski). Diese Nomenklaturen werden in einer Datenbasis geführt, die ständig automatisch auf dem neuesten Stand gehalten wird.

Mit der Angabe der Nomenklatur wählen wir aus, welches Gerät wir ansprechen wollen.

Property: Jedes Gerät ist jeweils einem bestimmten Gerätemodell zugeordnet (z.B. der Magnet `TK1MU1` dem Gerätemodell „Gepulste Magnete“, kurz MX). Alle Geräte eines Gerätemodells haben die gleichen Eigenschaften, ihre *Properties*. Diese Properties bestimmen, welche Möglichkeiten der Bedienung für ein Gerät bestehen, wie z.B. die Einstellbarkeit des Stroms bei Magneten, die Möglichkeit ein Gerät ein- oder auszuschalten, den Status zu lesen usw.

Es gibt zwei sogenannte *Kategorien* von Properties:

1. Die Master Properties. Das sind Properties, die immer das *gesamte* Gerät betreffen. Die Property `POWER` zum Ein- und Ausschalten eines Gerätes ist eine solche. Master Properties nehmen nicht an der Puls-zu-Puls-Modulation (PPM) des Beschleunigers teil, d.h. sie sind unabhängig von Events der Timingzentrale.
2. Die Slave Properties. Das sind Properties, die immer nur *einen virtuellen Beschleuniger* im Zyklus betreffen. So kann der Strom eines gepulsten Magneten für den Beschleuniger 0 geändert werden, ohne die Ströme des gleichen Magneten für die Beschleuniger 1 bis 15 zu beeinflussen. Slave Properties nehmen an der PPM des Beschleunigers teil. Das heißt, abhängig von den Events der Timingzentrale wird das Gerät, dem aktuellen virtuellen Beschleuniger entsprechend, neu eingestellt.

Zum Abschluss sei noch kurz auf Properties eingegangen, die *jedes* Gerät, unabhängig vom Gerätemodell, haben muss, weil es die Definition des Kontrollsystems so fordert. Es sind dies die Properties `POWER`, `STATUS`, `INFOSTAT`, `INIT`, `RESET`, `EQMERROR` und `VERSION` sowie für Geräte, die an der PPM teilnehmen noch `ACTIV` und `COPYSET`. Diese heißen *obligatorische* Properties.

Mit der Angabe der Property geben wir an, welche Eigenschaft des Gerätes wir nutzen wollen.

Property-Klasse: Zu den Properties gehören die Property-Klassen. Sie beschreiben die Art des Zugriffs auf die Property, also ob gelesen, geschrieben oder eine Funkton ausgeführt werden soll. Jede Kombination von Property mit zugehöriger Property-Klasse muss für jedes Gerät eindeutig sein.

Mit der Angabe der Property-Klasse geben wir an, ob wir vom Gerät einen Wert lesen oder am Gerät einen Wert setzen wollen, oder ob das Gerät einfach eine Funktion ausführen soll.

Virtueller Beschleuniger: Die Beschleunigeranlagen der GSI haben die Möglichkeit, quasi gleichzeitig bis zu 16 verschiedene Energien mit unterschiedlichen Teilchen zu beschleunigen bzw. zu führen. Dies wird mit der Puls-zu-Puls-Modulation (PPM) ermöglicht. Einen Puls, der für eine bestimmte Ionensorte mit einer bestimmten Energie verantwortlich ist, nennt man „Virtuellen Beschleuniger“. Greift man auf eine Slave Property zu, so muss die Nummer des gewünschten virtuellen Beschleunigers mit angegeben werden.

Damit wählen wir den virtuellen Beschleuniger aus, zu dem wir einen Wert schicken wollen, oder von dem wir einen Wert lesen wollen.

Werte: Wollen wir einen Schreibzugriff machen, also zum Beispiel einen Strom setzen, so muss natürlich auch der gewünschte Wert angegeben werden.

Die Zusammenfassung von Nomenklatur (der Name eines Gerätes), Property (eine der Eigenschaften des Gerätes) und Property-Klasse (die Angabe, ob gelesen, geschrieben oder eine Funktion ausgeführt werden soll) nennt man in NODAL ein *Datamodule*. Der Zugriff auf ein Gerät in dieser Art und Weise – wenn nötig mit Angabe des virtuellen Beschleunigers und eventuell einem Wert – heißt *Datamodule Call*.

Ein Datamodule Call ist die einfache Form der Gerätezugriffe in NODAL [2]. Im einfachsten Fall gibt man nur den Namen des Gerätes und die gewünschte Property an und kann schon Werte setzen oder lesen oder Aktionen ausführen lassen.

Bevor wir zu den ersten Gerätezugriffen kommen, wollen wir uns aber noch einige NODAL-Befehle ansehen, mit denen wir uns das oben gesagte noch etwas näher bringen können, und die uns auch später noch von Nutzen sein werden.

[2] Auf die andere Form der Zugriffe via Userface-Schnittstelle werden wir kurz am Ende eingehen.

4.2 Nützliche Befehle

Der Befehl `LISDM` listet uns alle Nomenklaturen der Geräte, die das Kontrollsystem kennt, auf. Dabei wird unterschieden zwischen den Geräten selbst und deren Kontrollrechnern (Gruppenmikros und Steuereinheiten (SE). Hinterlegte Nomenklaturen sind im Moment über das Kontrollsystem erreichbar.

```
>LISDM

 List of actually known devices:


TK2DG2_P TK3DM2_P TK3DW2_P SWTIME31 TK8DB2HP TK1MU1   S11MU1   UA_BN1
ECEMO3C  E02MK2   S12MB2   HFSQS2   S03KM2DV TS2VM3P  HHTMU3   XXXTEST1
...


 List of actually known computers:
ECMSMXPS XX2CGZZ_ TK0CG56_ TK0CS563 TK0CS564 TK0CS565 NODE01   S00CG70_
...
```

Das sind natürlich alle, also nahezu 3000 Geräte, die angezeigt werden. Will man nur eine Auswahl ganz bestimmter Geräte ansehen, dann gibt es den Befehl `LSTDM`. Diesem kann man Nomenklaturen mit sogenannten Wildcards mitgeben. Wildcards sind spezielle Zeichen, die als Ersatz für alle in NODAL zugelassenen Zeichen beliebiger Länge stehen. Die Wildcard in unserem Fall heißt „`*`“.

Wir möchten zum Beispiel alle Geräte am SIS aufgelistet haben. Dazu lassen wir uns alle Geräte auflisten, deren Nomenklatur mit „`S`“ beginnt, da alle Geräte am SIS (und nur diese) mit einem „`S`“ beginnen. Das geht einfach so:

```
>LSTDM("S*")

 List of actually known devices:
S12DC__P S12VP3I  S10KM3SS S11DX    S12DP    S12VV6H  S12KM3SN S12DP_I
...

 List of actually known computers:
S00CG61_ S00CS611 S00CS612 S00CG7F_ S00CS7F1 S00CS7F2 S00CS7F3 S00CS7F4
...
```

Oder alle Umlenker im Transferkanal:

```
>LSTDM("TK*MU*")

 List of actually known devices:
TK1MU1   TK1MU2   TK2MU3   TK4MU4   TK7MU5   TK7MU6

 List of actually known computers:

>
```

Will man von einem bestimmten Gerät wissen, zu welchem Gerätemodell es gehört und welche Properties es hat, benutzt man den Befehl `LSTNOMEN`, was soviel heißt wie „Liste die Datenbasiseinträge für die Nomenklatur xy“. Das geht wie folgt (wobei hier nicht die ganze lange Ausgabe gezeigt wird):

```
>LSTNOMEN("S12MU3I")

Nomenclature   : S12MU3I
Device typ     : TRAI
Device subtyp  : MAGN
Equipment model: MX_04
SIS address    : 007FH  [VME]
Device number  :   5

Property  PC  XSR  F   pc   pt   dc   dt  to  Exp  Unit
-------------------------------------------------------
CURRENTS  W     1  S    0   -     1   RF   6    0     A
CURRENTS  R     2  S    0   -     1   RF   6    0     A
CURRENTI  R     3  S    0   -     1   RF   6    0     A
...
CONSTANT  RA    7  M    0   -     4   RF   6    0     A
...
RESET     N    49  M    0   -     0   -    6    0     1
...

>
```

Hier können wir jetzt alles nötige herauslesen. Das Gerät `S12MU3I` gehört zum Gerätemodell `MX_04`, also zu den gepulsten Magneten. Es hat unter anderen die Property `CURRENTS` mit den Property-Klassen (`PC`) `W` für Schreibzugriffe und `R` für Lesezugriffe. Die Property-Kategorie (`F`) ist `S` für Slave-Property. Wie der Name der Property schon suggeriert, handelt es sich um den Strom des Magneten. Das angehängte „`S`“ dient zur Unterscheidung des Sollwertes (d. h. des einzustellenden Wertes) vom Istwert (d. h. des tatsächlich erzeugten Wertes), der über die Property „`CURRENTI`“gelesen werden kann.

Diese Funktionen werden wir immer einmal wieder brauchen, besonders dann, wenn Gerätezugriffe nicht funktionierten. Und Gerätezugriffe wollen wir nun kennenlernen.

Achtung! Achtung!! Achtung!!!

> Es sei *ausdrücklich* darauf hingewiesen, dass jegliches Ausprobieren der folgenden Beispiele zu Eingriffen an echten Geräten führt! Bevor nicht absolut sicher ist, dass der Strahl- und Experimentierbetrieb nicht beeinträchtigt wird, darf zu Übungszwecken *nicht* auf diese Geräte zugegriffen werden, sondern *nur* auf speziell eingerichtete Test- oder Dummy-Geräte!

4.3 Zugriffe auf Einzelwert-Properties

Eine Einzelwert-Property ist eine Property bei der nur genau *ein* Wert gesetzt oder gelesen werden kann. Mit dieser Art von Datamodules kann man in NODAL umgehen wie mit ganz normalen Variablen auch. Sie können links oder rechts von einer Zuweisung stehen und man kann mit ihnen rechnen.

Aber nun endlich los mit den ersten Gerätezugriffen.

4.3.1 Lesezugriffe

Zunächst ein Beispiel für einen einfachen Lesezugriff. Wir wollen den Istwert des Stroms des Magneten `TK1MU1` lesen. Wie wir dem Gerätemodell entnehmen, wird der Istwert mit der Property `CURRENTI` gelesen. Der Zugriff sieht dann einfach so aus:

```
>TYPE TK1MU1(CURRENTI)
    67.7682
```

Was wir eingetippt haben, ist der NODAL-Befehl `TYPE`, die Nomenklatur des Magneten (`TK1MU1`) und die gewünschte Property (`CURRENTI`) in Klammern direkt dahinter. Das Ergebnis ist ein Strom von rund 67.8 Ampere.

Noch ein weiterer Versuch. Wir wollen den Gerätestatus von `TK1MU1` lesen:

```
>TYPE TK1MU1(STATUS)
 -1089.0000
```

Die Formulierung ist wieder die gleiche, nur die Property `CURRENTI` wurde durch die Property `STATUS` ersetzt.

Nun wissen wir, dass beim Gerätestatus jedes einzelne Bit eine eigene Bedeutung hat, NODAL interpretiert aber zunächst einmal das ankommende Bitmuster als reele Zahl und liefert uns das entsprechende Ergebnis. Damit lässt sich natürlich nicht allzuviel anfangen. Wir möchten gern jedes Bit einzeln sehen und müssen das NODAL mitteilen. Dazu benutzen wir die Formatierung „`?`“ für die binäre Darstellung:

```
>TYPE ?TK1MU1(STATUS)
11111111111111111111101110111111
```

Zur Interpretation müssen wir nun noch wissen, dass die Bits von 0 bis 31 und zwar von rechts nach links zählen. In unserem Fall sind also die Bits 6 und 10 nicht gesetzt. Für Magnete des Gerätemodells MD_02 heißt das: Die Temperatur des Netzgerätes ist zu hoch (bit 10 = 0), was zu einem Hardwarefehler (Bit 6 = 0) führt.

Dem aufmerksamen Leser wird bereits aufgefallen sein, dass bei unseren bisher behandelten Beispielen die Angabe der Property-Klasse fehlt. Aber die ist bei Zugriffen auf Einzelwert-Properties, also auch bei den noch folgenden Beispielen, immer automatisch klar. Sagt man `TYPE`, dann will man einen Wert lesen. Sagt man `SET` und das Datamodule steht links vom Gleichheitszeichen, dann will man einen Wert setzen. Sagt man `SET` und das Datamodule steht rechts vom Gleichheitszeichen, dann will man einen Wert lesen.

Angenommen, wir wollen den gelesenen Status nicht gleich ausgeben, dann weisen wir den Status einer Variablen zu, die wir später weiter auswerten können, ohne jedesmal den Status vom Gerät neu lesen zu müssen:

```
>SET stat = TK1MU1(STATUS)
>TYPE ?stat
11111111111111111111101110111111
>TYPE "Status in Hex ist: " ]]stat
Status in Hex ist: FFFFFBBF
>
```

In unserem Beispiel steht das Datamodule rechts vom Gleichheitszeichen, also wird auch hier ein Wert gelesen (Property-Klasse „R“) und der Variablen `stat` zugewiesen. Den Wert dieser Variablen können wir dann einer weiteren Variablen zuweisen oder sie mit `TYPE` in verschiedenen Formaten ausgeben.

Bisher haben wir hier nur DC-Magnete, die nicht von Beschleuniger zu Beschleuniger umgetastet werden, also nicht an der PPM teilnehmen, behandelt. Deren Property `CURRENTI` ist eine Master-

Property. Aus diesem Grund mussten wir keinen virtuellen Beschleuniger berücksichtigen. Wollen wir einen Strom von einem gepulsten Magneten lesen, so müssen wir angeben, von welchem virtuellen Beschleuniger wir gerne den Strom hätten. In NODAL gibt es dazu die vordefinierte Variable `VRTACC` (für „Virtual Accelerator“). Ihr Wert gibt bei Slave-Properties an, für welchen virtuellen Beschleuniger der Geräteauftrag ausgeführt werden soll. Das gilt so lange für jeden Auftrag, bis der Wert der Variablen wieder geändert wird. Wir müssen `VRTACC` nur einmal richtig setzen.

Uns interessiere einmal der Istwert des Stroms des Magneten `TK9QD11` im Beschleuniger 8:

```
>TYPE VRTACC
     0.0000
>SET VRTACC = 8
>TYPE VRTACC
     8.0000
>TYPE TK9QD11(CURRENTI)
   165.6193
```

Nach dem Aufruf von NODAL hat `VRTACC` immer den Wert 0. Wird `VRTACC` vor einem Gerätezugriff nicht auf den gewünschten virtuellen Beschleuniger gesetzt, so arbeitet man immer mit Beschleuniger 0, und macht eventuell eine fertige Einstellung kaputt. Also, erhöhte Vorsicht, besonders bei Schreibzugriffen!

Hat man einmal den richtigen Beschleuniger gesetzt, so braucht man sich nicht mehr darum zu kümmern. Bei jedem Zugriff auf Slave-Properties ist jetzt immer der zuletzt gesetzte virtuelle Beschleuniger gemeint.

Zur Verdeutlichung wollen wir einmal von zwei verschiedenen Beschleunigern, aber vom selben Magneten, den Strom lesen:

```
>SET VRTACC = 5
>TYPE TK9QD11(CURRENTI)
   165.6193
>SET VRTACC = 11
>TYPE TK9QD11(CURRENTI)
      .0834
```

Obwohl die Formulierungen der beiden Datamodule Calls genau gleich sind, erhalten wir zwei verschiedene Stromwerte, weil wir aus zwei verschiedenen virtuellen Beschleunigern gelesen haben.

Weiter oben haben wir gesehen, wie es aussieht, wenn man sich den eingestellten virtuellen Beschleuniger ansieht:

```
>TYPE VRTACC
     0.0000
```

Wir wissen natürlich, das die virtuellen Beschleuniger nur die Nummern 0, 1, 2, usw. bis 15 annehmen können. Die Nachkommastellen bei der Ausgabe sind also etwas Fehl am Platze. Falls sie uns stören, veranlassen wir NODAL, mit einer Formatierung nur die Stellen vor dem Komma auszugeben:

```
>TYPE %2 VRTACC
 0
>SET VRTACC = 13
>TYPE %2 VRTACC
13
```

Unsere Formatierung ist „`%2`“ und sie besagt, dass die folgende Variable (hier `VRTACC`) als ganze Zahl mit zwei Stellen ausgegeben werden soll.

4.3.2 Schreibzugriffe

Ist das Gerät in Ordnung, kann man versuchen, einen Sollwert zu setzen. Wir wollen zunächst etwas vorsichtig sein und nur einen kleinen Strom setzen:

```
>SET TK1MU1(CURRENTS)=47.11
```

Im Unterschied zu einem Lesezugriff müssen wir hier natürlich noch zusätzlich einen Wert, nämlich den gewünschten Sollwert, mit angeben. Unser Datamodule steht diesmal links vom Gleichheitszeichen. Dadurch wird angezeigt, dass dies ein Schreibzugriff ist und NODAL liefert auch hier wieder automatisch die Property-Klasse `W` mit.

Nun können wir mit einem Lesezugriff schon testen, ob der Sollwert am Magneten angekommen ist und welcher Istwert tatsächlich eingestellt wurde:

```
>TYPE TK1MU1(CURRENTS)
    47.1133
>TYPE TK1MU1(CURRENTI)
    47.1219
```

Es ist wichtig, genau zwischen Sollwert lesen und Istwert lesen zu unterscheiden. Liest man den Sollwert (`CURRENTS`), so wird der (z.B. am Poti) *eingestellte* Wert zum Benutzer geschickt. Nur beim Lesen des Istwertes (`CURRENTI`) nimmt die Gerätesoftware tatsächlich eine Messung des Wertes vor, bevor auch dieser *gemessene* Wert zum Benutzer geschickt wird.

Unser erster Versuch ging gut. Das lässt uns für den zweiten etwas mutiger werden. Wir hoffen, dass die Kühlung gut funktioniert und setzen mal gleich 1000 Ampere.

```
>SET TK1MU1(CURRENTS)=1000

***  NODAL ERROR  76   Error in USR response

%MD-E-WCURRS_RANGE, USR 'W_CurrentS': Reference value out of range
>
```

Das ging schief. Unser Sollwert lag außerhalb des zulässigen Bereichs dieses Magneten (`Reference value out of range`). Die Gerätesoftware hat diesen Sollwert zurückgewiesen. Wie man diese Fehlermeldungen genau interpretiert, und wie man auch mit NODAL herausfindet, wie hoch der maximal zulässige Strom ist, wird später besprochen.

4.3.3 Rechnen

Natürlich (wie sollte es anders sein) existieren auch Properties, die nicht ganz in den Rahmen des Konzepts der Gerätebedienung passen. So gibt es z.B. bei Magneten des Gerätemodells `MD_02` eine Property `VOLTI`, die aus Kompatibilitätsgründen tut, was man nicht tun sollte. Sie liefert das Bitmuster eines ADCs anstatt einer physikalischen Größe zurück. Wir nehmen das zum Anlass, auf der Bedienungsebene aus diesem Bithaufen eine sinnvolle Größe zu machen. Was wir dazu wissen müssen ist die Skalierung. Also, welches Bitmuster entspricht dem maximalen Strom des Magneten? Das müssen wir bis jetzt noch durch Fragen oder Nachsehen herausfinden. Die Antwort ist: Bei 460 Ampere lesen wir $\texttt{7FFF}_{\texttt{hex}}$aus.

Zunächst sehen wir uns noch mal den aktuellen Wert mit Hilfe der Formatierung „`]]`“ hexadezimal an, also das Bitmuster selbst:

```
>TYPE ]]TK1MU1(VOLTI)
000034A3
```

Und nun eine kleine Rechnung, um den Wert in Ampère zu erhalten, wozu es zwei Möglich-

keiten gibt. Entweder wir rechnen zunächst den hexadezimalen Wert in einen dezimalen Wert um($7FFF_{hex} = 32767_{dez}$), oder wir geben ihn mit Hilfe einer weiteren Formatierung ([[ ) direkt an.

```
>TYPE TK1MU1(VOLTI) * 460 / 32767 " A"
   189.1690 A
>TYPE TK1MU1(VOLTI) * 460 / [[7FFF " A"
   189.1690 A
```

4.3.4 Lesen, Rechnen und Schreiben

Angenommen, uns interessiert der Wert des Strom eines Magneten garnicht, wir wollen ihn nur um fünf Prozent erhöhen, dann ist auch das ohne Probleme möglich:

```
>SET TK1MU1(CURRENTS) = 1.05 * TK1MU1(CURRENTS)
```

In *einer* Zeile wird der aktuelle Wert gelesen (Datamodule rechts vom Gleichheitszeichen!), um fünf Prozent erhöht (gerechnet) und der neue Wert zurückgeschrieben (Datamodule links vom Gleichheitszeichen!). Die richtige Reihenfolge der Gerätezugriffe (zuerst den aktuellen Wert lesen, dann den veränderten setzen) ist sichergestellt, da NODAL vor dem Setzen des Wertes diesen erst berechnen muss und dazu den aktuellen Wert liest.

4.4 Zugriffe auf Mehrwert-Properties

Eine Mehrwert-Property ist eine Property, bei der *mehr als ein* Wert mit *einem* Zugriff gelesen oder gesetzt wird. NODAL hat *keine* Möglichkeit, diese Werte genau so zu behandeln wie einen einzigen Wert bei den Einzelwert-Properties.

Aus diesem Grund müssen die Datamodule Calls für Mehrwert-Properties mit Hilfe des NODAL-Befehls `CALL` formuliert werden.

Benutzt man für einen Gerätezugriff die Formulierung mit `CALL`, so ist für NODAL klar, dass auf mehrere Variable gleichzeitig, also auf ein Feld, zugegriffen werden soll. Einzelwert-Properties lassen sich mit dieser Art der Datamodule Calls *nicht* ausführen!

4.4.1 Lesezugriffe

In einem Beispiel weiter oben haben wir versucht, den Strom des Magneten `TK1MU1` auf 1000 Ampere zu setzten, was zu einem Fehler führte, weil der Wert viel zu groß war. Wir mussten also herausfinden, wie groß der maximal zulässige Strom für `TK1MU1` sein darf. Schauen wir im Gerätemodell nach, so finden wir eine Property `CONSTANT`, die unter anderem auch den minimalen und maximalen Strom zurückliefert. Das Problem dabei ist, dass wir nicht einfach einen `TYPE`- oder `SET`-Befehl benutzen können, da ja `CONSTANT` immer *vier* Werte liefert und nicht einen, mit dem die Befehle `TYPE` und `SET` nur funktionieren.

Wir müssen uns zunächst also eine Variable konstruieren, die diese vier Werte aufnehmen kann. Wie wir bereits gesehen haben, hat man in NODAL dazu die Möglichkeit, ein Feld – eine Variable, die aus mehreren gleichartigen Variablen, den sogenannten Elementen, besteht – zu konstruieren. Wir wollen dieses Feld einmal `devcon` (für device constants – Gerätekonstanten) nennen und es so groß dimensionieren, dass genau unsere vier Werte hineinpassen:

```
>DIMENS devcon(4)
```

Nun können wir bereits den Gerätezugriff wagen, der jetzt – wie gesagt – mit dem NODAL-Befehl `CALL` formuliert wird.

Bei dieser Art der Formulierung ist es für NODAL nicht mehr ersichtlich, ob der Benutzer lesen oder schreiben möchte. Daher muss hier die Zugriffsart mit angegeben werden – `R` oder `r` für Lesen, `W` oder `w` für Schreiben.

Also los:

```
>CALL TK1MU1("R", devcon, CONSTANT)
```

Auch bei diesem Datamodule Call steht die Nomenklatur wieder vor und die Property in der Klammer. Hinzugekommen sind in der Klammer die Angabe der Zugriffsart und der Name der Feld-Variablen und vor dem Datamodule der Befehl `CALL` anstelle eines `SET` oder `TYPE`.

Verallgemeinert könnte man schreiben:

```
CALL <Nomenklatur>(<Zugriffsart>, <Feld-Variable>, <Property>)
```

Wir sagen hier „Zugriffsart“ und nicht „Property-Klasse“, weil in der Formulierung eines solchen Datamodule Calls die exakte Angabe der Property-Klasse zu einem Syntaxfehler führen würde. So darf zum Beispiel nicht `RA`, was die Property-Klasse ist, angegeben werden, sondern nur `R`, was wir Zugriffsart nennen, um einen Lesezugriff auf eine Mehrwert-Property zu machen. Das gleiche gilt für Schreibzugriffe. In allen anderen Fällen kann man die Begriffe „Zugriffsart“ und „Property-Klasse“ als synonym auffassen.

Und nun an die Auswertung. Wir wissen bereits, wie wir an die einzelnen Elemente unserer Variablen `devcon` herankommen können. Die einzelnen Elemente heißen genau wie unser Feld und sind von 1 beginnend durchnummeriert bis zur vorher von uns mit `DIMENS` festgelegten Größe, also bis 4. Nun heißt es erst mal etwas Tipparbeit:

```
>TYPE devcon(1)
     0.0000
>TYPE devcon(2)
   460.0000
>TYPE devcon(3)
     1.0000
>TYPE devcon(4)
     1.0000
```

Hat man nicht aufgepasst, dann geschieht folgendes:

```
>TYPE devcon(5)

***  NODAL ERROR  23   Array Dimension Error
```

NODAL soll auf ein Element zugreifen, das es gar nicht gibt. Das führt zu einem Fehler.

Diese (fehleranfällige) Tipparbeit können wir uns sparen. Das wird besonders interessant, wenn das Feld um einiges größer wird als unseres mit vier Elementen.

Wir haben bereits gesehen, dass es in NODAL die Möglichkeit gibt, eine Schleife zu konstruieren, die so oft durchlaufen wird, wie wir möchten. Damit können wir uns alle Werte einfach anzeigen lassen:

```
>FOR i = 1, 4; TYPE ! devcon(i)

     0.0000
   460.0000
     1.0000
     1.0000
>
```

Wir wollen uns zur Erinnerung noch mal ansehen, wie dieser Befehl funktioniert. Die sogenannte Laufvariable `i` nimmt nach und nach die Werte von 1 bis 4 an. Für jeden Wert von `i` werden die Befehle nach dem Semikolon durchlaufen. Zunächst für `i = 1` einen Zeilenvorschub (`!`) und die erste Variable – also devcon(1), weil ja als Elementnummer in `devcon(i)` wieder unsere Laufvariable `i` erscheint – ausgeben. Dann für `i = 2` usw. bis auch devcon(4) ausgegeben ist. Danach wird die Schleife automatisch verlassen.

Um zu sehen, wie sich die Laufvariable `i` ändert und um die Ausgabe etwas schöner zu machen, wollen wir unsere `FOR`-Schleife noch etwas ausbauen:

```
>FOR i = 1, 4; TYPE ! "devcon(" %1 i ") = " %4 devcon(i)

devcon(1) =    0
devcon(2) =  460
devcon(3) =    1
devcon(4) =    1
>
```

Zunächst kommt wieder der Zeilenvorschub. Die beiden in Gänsefüßchen eingeschlossenen Zeichenketten „`devcon(`“ und „`) = `“ werden genau so ausgegeben wie sie sind. Dazwischen, also quasi in die Klammer, kommt die Laufvariable `i` mit einer Stelle (`%1`) und dahinter unsere Variable selbst mit vier Stellen (`%4`) .

Zuletzt wollen wir noch probieren, was passiert, wenn man versucht, eine Mehrwert-Property mit einer Formulierung für Einzelwert-Properties zu lesen.

```
>TYPE TK1MU1(CONSTANT)

***  NODAL ERROR  75   Error during USERFACE call
%UFC-E-NOMATCH_PROPERTY, dbs_access failed when searching property and
p_class
>
```

Nun, es war zu erwarten, dass das nicht gut geht. Wir wollen einmal versuchen, die Fehlermeldungen zu interpretieren.

Zunächst einmal die Meldung „`*** NODAL ERROR 75 ...`“. NODAL sagt, dass ein Fehler während eines Zugriffs auf das Userface auftrat, obwohl wir Userface gar nicht aufgerufen, sondern eindeutig einen Gerätezugriff via Datamodule Call gemacht haben. Diese Meldung wird aber sofort einsichtig, wenn man weiß, dass jede Formulierung eines Gerätezugriffs via Datamodule Call von NODAL intern in einen Aufruf des Userface umgesetzt wird. Auch für NODAL gibt es also keinen anderen Zugriff auf Geräte als über das Userface.

Somit ist klar, dass die Meldung `%UFC-E-NOMATCH...` vom Userface (`%UFC`) kommt. Dieses hat auf die Datenbasis zugegriffen (`dbs_access`), um zu sehen, ob es für den Magneten `TK1MU1` die angegebene Property mit Property-Klasse gibt. Es müßte also einen Eintrag `CONSTANT` für die Property geben und – wie wir bereits wissen, für diese Art der Zugriffe ist die Property-Klasse automatisch klar – einen Eintrag „`R`“ (für Read – lesen eines Einzelwertes) für die Property-Klasse.

Das können wir nachsehen mit dem bereits bekannten NODAL-Befehl `LSTNOMEN`:

```
>LSTNOMEN("TK1MU1")

Nomenclature   : TK1MU1
Device typ     : MAGN
...

Property  PC  XSR  F   pc   pt   dc   dt  to  Exp  Unit
-----------------------------------------------------
...
CONSTANT  RA    7  M    0   -     4   RF   6    0     A
...

>
```

Also, die Property `CONSTANT` gibt es zwar für unseren Magneten, aber sie hat die Property-Klasse (`PC`) „Read Array“ (`RA` = Lese Feld) und nicht „Read“ (`R`). Aus diesem Grund bekamen wir die Fehlermeldung „`%UFC-E-NOMATCH...`“, weil die Property-Klasse „`R`“ für die Property `CONSTANT` nicht gefunden wurde.

4.4.2 Schreibzugriffe

Schreibende Zugriffe auf Mehrwert-Properties funktionieren genauso wie die Lesezugriffe. Man muss eben nur die Zugriffsart „`W`“ angeben und die zu schreibenden Daten natürlich vorher festlegen. Als Beispiel wählen wir diesmal einen der Bumper, weil diese im Gerätemodell, im Gegensatz zu den Magneten, eine Mehrwert-Property mit Schreiberlaubnis haben (mit `LSTNOMEN` nachzuprüfen):

```
>TYPE %2 VRTACC
11
>SET VRTACC = 15
>DIMENS BumpVar(2)
>SET BumpVar(1) = 0815
>SET BumpVar(2) = 4711
>CALL S11MB1("W", BumpVar, PARA)
```

Auch hier haben wir wieder erhöhte Vorsicht gelten lassen und daran gedacht, dass bei Slave-Properties der richtige virtuelle Beschleuniger (`VRTACC`) vor dem Gerätezugriff gesetzt sein muss. Unser Feld, das wir auf die Größe 2 dimensionieren, heißt diesmal `BumpVar`. `BumpVar(1)` und `BumpVar(2)` sind die Elemente des Feldes, deren Werte zum Bumper geschickt werden sollen. `PARA` ist der Name der Mehrwert-Slave-Property der Bumper. Mit unserem Datamodule Call werden die Werte `815` und `4711` zum Bumper `S11MB1` geschickt.

4.5 Zugriffe auf Properties ohne Datenwerte

Wie man mit dem Befehl `LSTNOMEN` leicht nachprüfen kann, gibt es auch Properties, die Daten weder schreiben noch lesen können. Diese Properties mit der Property-Klasse „`N`“ sind dazu da, um eine *Funktion* auszuführen. So geht das viel gebrauchte Reset nicht nur per Knopfdruck an der SE, sondern auch per NODAL von einem Terminal aus:

```
>CALL TK1MU1(RESET)
```

Dies ist die dritte Art der Formulierung eines Datamodule Calls. Auch in dieser Art der Formulierung wird der `CALL`-Befehl benutzt. Das wichtigste Merkmal hierbei ist, dass keinerlei Daten

beim Gerätezugriff behandelt werden. Die Feld-Variable, die wir bei den Mehrwert-Properties kennengelernt haben, fällt bei diesem `CALL`-Befehl einfach weg. Und auch die Property-Klasse wird weggelassen. Ebenso fehlt in dieser Formulierung eine Zuweisung mit Gleichheitszeichen wie bei den Einzelwert-Properties. Damit ist bei der Formulierung

```
>CALL <Nomenklatur>(<Property>)
```

die Property-Klasse wieder automatisch klar, nämlich „N“.

4.6 Zugriffe auf Properties mit Parametern

Parameter werden benutzt, um einen Gerätezugriff genauer zu spezifizieren. So kann ein Parameter zum Beispiel dazu benutzt werden, einem Gerät mitzuteilen, welche Daten ihm geschickt werden oder welche Daten man gerne lesen möchte. Oder man benutzt Parameter, um der Gerätesoftware mitzuteilen, wie sie einen Gerätezugriff genau zu behandeln hat.

Das heißt also, dass Parameter dem Auftrag an ein Gerät zusätzlich zu den üblichen Dingen mitgegeben werden müssen.

4.6.1 Einzelwert-Properties mit Parametern

Für einfache Beispiele mit Einzelwert-Properties gibt es bei GSI leider (noch) kein passendes Gerät. Wir basteln uns also schnell ein Phantasiegerät und nennen es `XY_DEV`. Dieses soll eine Property `PROP` haben, die wahlweise, mit einem Parameter gesteuert, einen oder einen anderen Wert setzen oder lesen kann.

Mit Parameter = 0 setzen oder lesen wir den einen Wert, mit Parameter = 1 den anderen:

```
>SET XY_DEV(PROP(0)) = 47.11
>SET XY_DEV(PROP(1)) = 0.815
>TYPE XY_DEV(PROP(0))
    47.1100
>TYPE XY_DEV(PROP(1))
      .8150
```

Der einzige Unterschied zur Formulierung eines Zugriffs auf eine Einzelwert-Property ohne Parameter ist eben dieser zusätzliche Parameter. Er wird in Klammern direkt hinter der Property angegeben.

4.6.2 Mehrwert-Properties mit Parametern

Besser gestellt, wenn es um reale Beispiele geht, sind wir bei Mehrwert-Properties. Die gerampten Magnete brauchen so viele Stromwerte für ihre Rampen, dass diese Werte (noch) gar nicht mit einem einzigen Paket zum Gerät geschickt oder vom Gerät gelesen werden können. Zwei Parameter, die in einem Teilpaket mitgeschickt werden, beschreiben, zu welchem Abschnitt der Rampe die Daten gehören.

Wir wollen 100 Werte in einem Teilpaket schicken, in dem sich die Daten vom 300sten bis zum 399sten Rampenwert befinden. Es soll also ein Feld geschickt werden. Deshalb müssen wir die Formulierung für Mehrwert-Properties benutzen. Wir schicken also die 100 Daten und teilen dem Gerät gleichzeitig mit, dass die Daten zum Rampenabschnitt 300 bis 399 gehören:

```
>DIMENS i_ramp(100)
>SET i_ramp(1) = 48.45
>SET i_ramp(2) = 49.17
...
>SET i_ramp(100) = 98.77
>CALL S11MU1("W", i_ramp, CYCRMPSC(300, 399))
```

Auch hier werden die Parameter wieder in Klammern direkt hinter der Property angegeben, die hier `CYCRMSPC` heißt. Da es zwei sind, werden sie durch ein Komma getrennt.

Das Zurücklesen der Daten funktioniert dann ganz analog. Zusätzlich haben wir die beiden Parameter diesmal als Variablen angegeben:

```
>SET par1 = 300
>SET par2 = 399
>CALL S11MU1("R", i_ramp, CYCRMPSC(par1, par2))
>FOR i = 1, 100; TYPE ! "i_ramp(" i + 299 ") = " i_ramp(i)

i_ramp(300.0000) =   48.4537
i_ramp(301.0000) =   49.1674
...
i_ramp(399.0000) =   98.7721
>
```

4.6.3 Properties ohne Datenwerte mit Parametern

Steuerungen von Funktionen mit Parametern funktionieren ebenso. Auch hier fehlt uns wieder ein konkretes Beispielgerät. Sei also unser Phantasiegerät in der Lage, zwei unterschiedliche Reset-Funktionen auszuführen:

```
>CALL XY_DEV(RESET(0))
```

Für Einzelwert-, Mehrwert- und Properties ohne Datenwerte gilt also ganz allgemein: Parameter werden in Klammern und, falls mehrere benötigt werden, durch Kommas getrennt direkt hinter der Property, die Parameter erwartet, angegeben.

5 Wie kommt ein Sollwert an sein Ziel

Für die Spezialisten, die es interessiert, was hinter einem Gerätezugriff eigentlich steckt, wollen wir noch einen kleinen Ausflug in die Tiefen des Kontrollsystems wagen. Wir wollen uns ansehen, wie ein Sollwert, der von NODAL aus abgeschickt wird, an sein Ziel kommt und woher das Kontrollsystem weiß, wo dieses Ziel eigentlich ist.

Dazu formulieren wir schnell einen Schreibzugriff auf ein Gerät

```
>SET VRTACC = 5
>SET S12MU3I(CURRENTS)=17.4
```

und schauen uns das Gerät selbst noch einmal genauer an:

```
>LSTNOMEN("S12MU3I")

Nomenclature   : S12MU3I
Device typ     : TRAI
Device subtyp  : MAGN
Equipment model: MX_04
SIS address    : 007FH  [VME]
Device number  :   5

Property  PC  XSR  F   pc   pt   dc   dt  to  Exp  Unit
-------------------------------------------------------
CURRENTS  W     1  S    0   -     1   RF   6    0     A
CURRENTS  R     2  S    0   -     1   RF   6    0     A
CURRENTI  R     3  S    0   -     1   RF   6    0     A
...

>
```

Alle diese Angaben von `LSTNOMEN` sind Einträge in der Datenbasis. Diese Einträge werden nun dazu benutzt, um den Auftrag auf Korrektheit zu überprüfen, ihn an die richtige Stelle im Kontrollsystem zu schicken und dort die weitere Verarbeitung zu veranlassen.

Also los: `S12MU3I` hat eine Property `CURRENTS` und diese eine Property-Klasse „`W`“ für Schreiben. Es wird nur ein einzelner Datenwert (`dc` = data count = 1) und keine Parameter (`pc` = parameter count = 0) erwartet. Der virtuelle Beschleuniger wurde auch gesetzt, somit ist unsere Auftrags*formulierung* also korrekt.

Dem Magneten ist die `SIS address` 7F zugeordnet. Das heißt, dass der Magnet am VME-Rahmen mit der Ethernet-Knotennummer 7F hängt. Damit wissen wir schon, wo unser Auftrag hin muss. Nun hängen natürlich viele Geräte an einem VME-Rahmen. Diese unterscheiden sich durch ihre unterschiedlichen Gerätenummern. `S12MU3I` hat die `Device number` 5. Und somit kennen wir unser endgültiges Ziel. Die Nomenklatur des Magneten wurde in eine eindeutige, vom Kontrollsystem zu verarbeitende Addresse übersetzt.

Nun ist zwar die Post angekommen, es muss aber noch geklärt werden, was mit den Daten, die in dem Paket stecken, zu geschehen hat. Für jede Property und Property-Klasse gibt es vor Ort, also auf VME-Ebene, ein Stück Software, das die weitere Bearbeitung des Benutzerauftrags, die Bedienung des Geräts, übernimmt. In unserem Fall ist es die USR[3] (es gibt auch noch SSRs[4], deshalb `XSR` in der Überschrift) mit der Nummer 1, die die weitere Bearbeitung des Auftrags übernimmt. Somit ist die Property in eine vom Kontrollsystem zu verarbeitende Vorschrift übersetzt.

[3] USR – User Service Routine
[4] SSR – System Service Routine

Kommt jetzt innerhalb von 6 Sekunden (`to` = timeout = 6) eine positive Antwort – die Software auf der VME-Ebene weiß, wo die Antwort hinzuschicken ist, denn wir haben einen Absender auf dem Paket angegeben –, so ist unser Auftrag erfolgreich ausgeführt worden.

So funktioniert im Prinzip jeder Gerätezugriff, sei es ein Schreib-, Lese- oder Funktionszugriff.

6 Fehlermeldungen

Im Grunde kann man die Fehlermeldungen, die bei der Benutzung von NODAL auftreten können, in drei Kategorien aufteilen.

6.1 NODAL-Fehlermeldungen

Diese Art haben wir schon kennengelernt. Es sind Meldungen über Fehler, die in NODAL selbst aufgetreten sind (z.B. Syntaxfehler, falsche Zuweisung von Variablen usw.). Tritt ein Fehler im Userface oder in der Gerätesoftware auf, so führt das ebenfalls (zusätzlich) zu einer Fehlermeldung von NODAL, die nichts anderes besagt, als dass im Userface oder in der Gerätesoftware ein Fehler erkannt wurde. Das sieht zunächst etwas „doppelt gemoppelt“ aus, ist aber sehr nützlich, wenn man NODAL im Program-Modus benutzt, worauf wir später noch eingehen werden.

Alle Fehlermeldungen von NODAL haben die folgende Form:

```
***  NODAL ERROR <Errornummer>   <beschreibender Text>
```

Die möglichen Fehlermeldungen von NODAL sind im Anhang ab Seite 75 aufgelistet.

6.2 Fehlermeldungen vom Userface

Ist ein Gerätezugriff in NODAL korrekt formuliert, wird er von NODAL vorverarbeitet und anschließend an das Userface übergeben. Dieses führt weitere Tests durch. Es wird zum Beispiel geprüft, ob das anzusprechende Gerät die angegebene Property hat, ob das Gerät erreichbar (online) ist, ob die Angabe der Werte richtig ist und vieles mehr. Wird bei diesen Tests ein Fehler festgestellt, so wird *kein* Gerätezugriff durchgeführt. Userface meldet NODAL einen Fehler.

Alle Meldungen von Userface haben die folgende allgemeine Form:

```
%UFC-<Schwere>-<Kennung>, <beschreibender Text>
```

Die Schwere gibt an, wie schlimm der Fehler ist. „`W`“ (= Warning) ist eine Warnung. „`E`“ (= Error) ist ein „normaler“ Fehler. Diese Art tritt am allerhäugfigsten auf. „`F`“ (= Fatal) ist ein schwerer oder fataler Fehler. Außer diesen drei Schweren von Fehlern gibt es noch zwei verschiedene Erfolgsmeldungen. Es sind dies „`S`“ (= Success = Erfolg) und „`I`“ (= Information), die normalerweise nicht angezeigt werden.

Die Kennung ist eine manchmal leider kaum noch entzifferbare Kurzbezeichnung für den aufgetreten Fehler.

Der beschreibende Text erklärt den Fehler in kurzen Worten.

Ein konkretes Beispiel ist die folgende Meldung:

```
%UFC-E-DEVICE_OFFLINE, an addressed device is off-line
```

Tritt ein Fehler in Userface auf, so ist die zugehörige Fehlermeldung von NODAL:

```
***  NODAL ERROR  75   Error during USERFACE call
```

Eine weitere Fehlermeldung des Userface haben wir bereits auf Seite 28 kennengelernt.

6.3 Fehlermeldungen von der Gerätesoftware

Wurden auch im Userface alle Tests erfolgreich bestanden, so führt dieses einen Gerätezugriff durch. Mit anderen Worten, Userface schickt einen Auftrag an einen VME-Knoten, der diesen Auftrag weiterbearbeitet. Tritt während dieser Bearbeitung ein Fehler auf, so meldet die Gerätesoftware an Userface und dieses an NODAL einen Fehler.

Alle Meldungen der Gerätesoftware haben die folgende allgemeine Form:

```
%<Geraetemodell>-<Schwere>-<Kennung>, <beschreibender Text>
```

Schwere, Kennung und beschreibender Text haben die gleiche Bedeutung wie bei den Userface-Fehlern. Gerätemodell ist normalerweise die Kurzbezeichnung der verschiedenen Gerätemodelle. Für DC-Magnete „`MD`“, für die Kicker „`MK`“ und so weiter. Ausnahmen bilden die Meldungen der *System*software auf der VME-Ebene. Diese beginnen mit „`ECM`“, wenn sie von der SE kommen oder mit „`XSR`“, wenn sie vom Gruppenmikro kommen.

Konkrete Beispiele sind die folgenden Meldungen:

```
%MX-E-CurrS_Sequence, EQM 'CurrentS' detected event sequence error
%MD-E-NoInverter, USR 'W_Inverter': Device has no inverter
%ECM-E-MILSD-TIMEOUT, Timeout detected during MIL-SD-communication
%XSR-E-DEV_OFFLINE, addressed device is offline
```

Die erste Meldung (`%MX-E-...`) kommt von dem Equipment Module (EQM) CurrentS. Das ist der Teil der Gerätesoftware auf der SE, der für das Setzen des Strom-Sollwertes verantwortlich ist. Diese Software hat festgestellt, dass die Sequenz der für sie relevanten Events nicht in Ordnung ist. Entweder fehlen Events, oder sie sind nicht in der richtigen Reihenfolge. Die zweite Meldung (`%MD-E-...`) kommt von der User Service Routine (USR) W_Inverter. Das ist der Teil der Gerätesoftware auf der Ebene des Gruppenmikros, der für das Umschalten des Polwenders zuständig ist. Dieser Magnet besitzt aber keinen Polwender. Die dritte Meldung (`%ECM-E-...`) kommt vom Equipment Control Monitor (ECM). Das ist die Systemsoftware, die auf der SE läuft. Diese hat einen Fehler während der Kommunikation mit der Interfacekarte festgestellt. Die letzte Meldung (`%XSR-E-...`) kommt von einer System Service Routine (SSR). Das ist ein Teil der Systemsoftware auf der Ebene des Gruppenmikros. Diese hat festgestellt, dass das anzusprechende Gerät im Moment nicht erreichbar ist.

Tritt ein Fehler in der Gerätesoftware auf, so ist die zugehörige Fehlermeldung von NODAL:

```
**  NODAL ERROR  76   Error in USR response
```

Eine weitere Fehlermeldung der Gerätesoftware haben wir bereits auf Seite 25 kennengelernt.

Nun sind gerade Fehlermeldungen der Gerätesoftware oft auch dann noch nicht so recht verständlich, wenn man den beschreibenden Text gelesen hat, zumal er in englisch ist. Dazu gibt es mittlerweile für einige (wenige) Gerätemodelle weitere Hilfen, um herauszufinden, welches Problem hinter einer Fehlermeldung verborgen ist. Mit dem HELP-Kommando auf der VAX-Ebene (*nicht*, wenn man in NODAL ist!) bekommt man eine, wenn nötig, ausführlichere und vor allem deutsche Erklärung des aufgetretenen Fehlers. Dazu muss man das Gerätemodell und die aufgetretene Fehlermeldung (ohne den beschreibenden Text) eingeben.

Angenommen, wir wollen zu der Meldung

```
%MX-E-CurrS_Sequence, EQM 'CurrentS' detected event sequence error
```

etwas genaueres wissen. So tippen wir einfach folgendes (auf VAX-Ebene) ein:

```
A $ HELP MX MESSAGES MX-E-CURRS_SEQUENCE
```

Das Ergebnis ist dann die folgende Ausgabe:

```
MX

  Messages

    MX-E-CurrS_Sequence
        Message Text EQM 'CurrentS' detected event sequence error
        Description  Die fuer dieses Geraet relevanten Events sind
                     in einem virtuellen Beschleuniger in der falschen
                     Reihenfolge oder unvollstaendig.
```

Wir wollen hier nicht weiter auf das VAX-HELP eingehen, weil das zu weit von unserem Thema wegführen würde. Interessierte sollten einfach einmal fragen, nachlesen oder auf VAX-Ebene

```
A $ HELP HELP
```

eingeben.

7 Programm-Modus

Bisher wurde nur der Direkt-Modus behandelt, bei dem alle Befehle sofort nach dem Eingeben ausgeführt werden. Bei den meisten auftretenden Problemen erfordert das entsprechend viel Tipparbeit, die jedesmal wiederholt werden muss, wenn die gleiche Aktion noch einmal durchzuführen ist.

Um sich die Arbeit zu erleichtern, kennt NODAL auch den Programm-Modus. Hierbei werden Sequenzen von Befehlen der Reihe nach abgearbeitet. Natürlich müssen sie einmal eingegeben werden, können aber danach beliebig oft abgerufen werden.

So gut wie alle Befehle, insbesondere die bisher behandelten, können sowohl im Direkt-Modus als auch im Programm-Modus verwendet werden.

7.1 Aufbau der Programme

Programme in NODAL errinnern sehr an BASIC-Programme. Sie sind zeilenweise aufgebaut, wobei alle Zeilen am Zeilenanfang durch eine Nummer gekennzeichnet sind. Ein Befehl wird (unter anderem) durch das Ende einer Zeile abgeschlossen. Länger als eine Zeile (79 Zeichen) darf ein Befehl nicht sein, es sind also *keine Folgezeilen* vorgesehen. Dagegen ist es aber möglich, mehrere Befehle in eine Zeile zu schreiben. Die Zeilen werden automatisch nach aufsteigenden Zeilennummern sortiert und in dieser Reihenfolge abgearbeitet. Die Zeilennummern müssen dabei nicht fortlaufend sein.

Im Unterschied zu BASIC sehen die Zeilennummern allerdings zunächst etwas merkwürdig aus. Sie bestehen aus zwei ein- oder zweistelligen Zahlen, die durch einen Punkt getrennt sind. Dabei ist die Null jeweils nicht erlaubt. Die Bedeutung dieser beiden Anteile wird später noch näher erläutert. Zunächst ist nur wichtig, dass beim Sortieren der Zeilennummern zunächst die Zahlen vor dem Punkt betrachtet werden und, wenn diese gleich sind, die nach dem Punkt (es wird also sortiert, als ob die Zeilennummern reelle Zahlen sind).

Die gültigen Zeilennummern sind, schon sortiert:

```
 1.01
 1.02
  ⋮
 1.99

 2.01
 2.02
  ⋮
 2.99

 3.01
  ⋮
99.98
99.99
```

In einer NODAL-Zeile muss an erster Stelle nach der Zeilennummer ein NODAL-Befehl stehen. Dazwischen sind nur Leerzeichen erlaubt.

7.2 Programme eingeben

Die einfachste Möglichkeit, Programme einzugeben, besteht darin, sie zeilenweise einzutippen. So stellt die folgende Eingabe bereits ein kleines NODAL-Programm dar:

```
>10.10 SET Wert=17
>10.20 TYPE Wert+4
>
```

Welche Zeilennummer zuerst eingegeben wird, ist ohne Bedeutung, sie werden sowieso automatisch aufsteigend sortiert. Wie schon erwähnt, braucht die Nummerierung nicht fortlaufend zu sein. Es empfiehlt sich sogar, die Nummern so zu wählen, dass jeweils noch einige Befehle dazwischen passen: Dann kann man, falls erforderlich, noch etwas einfügen, ohne alle Zeilen neu nummerieren zu müssen.

Gibt es eine Zeilennummer bereits, wird die alte Zeile gelöscht und statt dessen die neu eingegebene verwendet. Durch die Eingabe der Zeilennummer allein, also ohne Befehl dahinter, wird die entsprechende Zeile gelöscht. Das kann benutzt werden, um Zeilen in NODAL zu ändern. Allerdings gibt es dafür bessere Verfahren, auf die noch eingegangen wird.

Nachdem das Programm eingegeben wurde, befindet es sich im NODAL-Arbeitsspeicher.

7.3 Programme ausführen

Nachdem man ein Programm eingegeben hat, möchte man es auch einmal ausführen. Dazu dient in NODAL der Befehl `RUN`. Er startet das Programm, dass sich gerade im NODAL-Speicher befindet, d.h. die Befehle werden Zeile für Zeile abgearbeitet.

Nachdem das oben erwähnte Programm eingegeben wurde, würde der Befehl `RUN` also bewirken, dass der Variablen `Wert` der Wert 17 zugewiesen wird (Zeile 10.10) und danach die Summe von `Wert` und `17`, also 21, ausgegeben wird (Zeile 10.10):

```
>RUN
    21.0000
>
```

Der Befehl `RUN` kann abgekürzt werden bis auf `RU`.

7.4 Fehler bei der Programmausführung

Wird bei der Ausführung eines Programms ein Fehler festgestellt, bricht NODAL die Ausführung des Programms ab und gibt eine Fehlermeldung aus, die der im Direkt-Modus entspricht. Zusätzlich wird noch die Nummer der Zeile angezeigt, in der der Fehler aufgetreten ist.

Als Beispiel wird ein Programm gezeigt, in dem versucht wird, durch Null zu teilen:

```
>1.10 SET x=1/0
>RUN
***  NODAL ERROR  6    Attempt to Divide By Zero
At line:  1.10  in  Main
>
```

Entsprechend zum Direkt-Modus lässt sich durch `Ctrl-B` (die Tasten `CTRL` und `B` gleichzeitig drücken) die Fehlerstelle anzeigen. Es wird die Zeile mit dem Fehler angezeigt, wobei die Schreibmarke auf die Stelle zeigt, an der der Fehler festgestellt wurde. Das ist nicht immmer die eigentlich falsch Stelle, aber doch zumindest ein guter Hinweis auf den Fehler. Die angezeigte

Zeile kann direkt korrigiert und durch Drücken der RETURN-Taste in das Programm übernommen werden.

7.5 Anzeigen eines Programmes

Um zu sehen, was man programmiert hat bzw. welches Programm sich gerade im Arbeitsspeicher befindet, ist der Befehl LIST vorgesehen. Nach der Eingabe des Befehlswortes, das bis auf LI abgekürzt werden kann, wird das gerade im Arbeitsspeicher befindliche Programm auf dem Bildschirm dargestellt.

Nachdem das behandelte Beispiel eingegeben wurde, würde sich ergeben:

```
>LIST
10.10 SET Wert=17
10.20 TYPE Wert+4
>
```

Nachdem ein Programm mindestens einmal abgearbeitet wurde, ergibt sich allerdings ein etwas anderes Bild. Alle Zeilen, die ausgeführt wurden, werden durch einen Klammeraffen (@) gekennzeichnet. Weiter komprimiert NODAL die Zeilen, indem alle nicht erforderlichen Leerzeichen beseitigt werden und von allen abkürzbaren Befehlen die jeweils kürzeste Form angezeigt wird. Nachdem einmal der Befehl RUN aufgerufen wurde, würde sich also ergeben:

```
>LIST
10.10 @SE Wert=17
10.20 @T Wert+4
>
```

Die Kennzeichnung durch den Klammmeraffen kann manchmal ganz hilfreich sein, um zu sehen, wie weit ein Programm abgearbeitet wurde.

Besonders bei umfangreichen Programmen ist oft ein Ausdruck auf Papier wünschenswert. Dazu gibt es in NODAL den Befehl FLIST. Mit ihm wird das gerade im Arbeitsspeicher befindliche Programm in dem aktuellen Dateiverzeichnis („directory“) unter dem Namen NODAL_MAIN.LIS abgelegt und kann z.B. anschließend ausgedruckt werden. Es ist auch möglich, die Auflistung unter einem anderen Namen abzulegen, indem der Name der Datei (wobei auch Knotenkennungen, Dateiverzeichnisse usw. möglich sind) als Parameter angegeben wird. So wird das Programm in der Datei MEIN_PROGRAMM.V17 abgelegt durch

```
>FLIST("MEIN_PROGRAMM.V17")
>
```

Einen sofortigen Ausdruck erhält man, wenn als Dateiname ein Drucker angegeben wird[5].

7.6 Speichern

Beim Beenden von NODAL werden Programme, die sich eventuell im Arbeitsspeicher befinden, nicht automatisch irgendwo gesichert — sie sind nach dem Verlassen von NODAL weg. Besonders bei umfangreichen Programmen wäre es sehr mühsam, sie jedesmal wieder neu eintippen zu müssen. Daher besteht in NODAL die Möglichkeit, Programme in einer Datei abzuspeichern und sie davon wieder einzulesen.

Die Programme werden unter einem Namen abgelegt. Dieser Name darf aus Buchstaben, Ziffern, Dollarzeichen ($) und dem Unterstrich (_) bestehen. Die Länge darf praktisch beliebig sein, solange

[5] Auf dem Rechner *ALICE* wird das Programm durch FLIST("LPA0:"), auf den HKR-Rechnern durch FLIST("A::LPA0:") auf dem Drucker im Rechnerraum ausgegeben.

der Name noch in eine Bildschirmzeile passt (das sind 80 Zeichen).

Bei dem Namen handelt es sich eigenlich um einen VMS Dateinamen, bei dem auch Dateiverzeichnisse („directory“) und eine Namens-Erweiterung („extension“) angegeben werden können. Fehlen diese, werden sie von NODAL automatisch mit Standardwerten ergänzt, so dass sie normalerweise weggelassen werden können.

Beachtet werden muss, dass Programme für den Rechner *ALICE* und die HKR-Rechner getrennt abgespeichert werden. Will man auf den jeweils anderen Rechner zugreifen, ist die Knotenkennung voranzustellen. Soll von dem Rechner *ALICE* auf Programme auf den HKR-Rechnern zugegriffen werden, ist den Namen die Kennung `A::` voranzustellen. Umgekehrt ist die Kennung `C::` voranzustellen, wenn von dem Rechner *ALICE* auf Programme der HKR-Rechner zugegriffen werden soll[6].

7.6.1 Abspeichern

Zum Abspeichern ist der Befehl `SAVE` vorgesehen. Hinter dem Befehlswort `SAVE` ist der Name anzugeben, den das Programm beim Abspeichern bekommen soll. Der Befehl kann bis auf `SA` abgekürzt werden.

So würde durch

```
>SAVE BEISPIEL
>
```

ein im Arbeitsspeicher befindliches Programm unter dem Namen `BEISPIEL` abgelegt.

Bei der Wahl des Namens ist zu bedenken, dass es in der GSI viele Benutzer von NODAL gibt. Verwenden nun zwei den gleichen Namen, ist das erste Programm nicht mehr ohne weiteres zugänglich, da unter dem Namen das zuletzt abgespeicherte Programm angesprochen wird. (Falls eine doppelte Namensgebung aus Versehen passiert: Eine zeitlang sind die anderen Programme unter dem gleichen Namen noch vorhanden und können auch wieder gelesen werden. Aber ab und an werden sie wirklich gelöscht, um Speicherplatz zu sparen, und sind dann weg.)

Daher ist, wenn man sich nicht sicher ist, ob ein Name schon von anderen Benutzern schon verwendet wurde, anzuraten, z.B. das Namenskürzel im Namen zu verwenden. Ein Benutzer Xenakis Zysk mit dem Kürzel XZ würde also sein Beispiel besser als

```
>SAVE BEISPIEL_XZ
>
```

abspeichern.

7.6.2 Einlesen

Entsprechend zum Befehl `SAVE` gibt es auch Befehle zum Wiedereinlesen eines abgespeicherten Programms. Der wichtigste davon lautet `OLD` , abkürzbar bis auf `OL`. Hinter dem Befehlswort ist der Name anzugeben, unter dem das Programm mit dem Befehl `SAVE` abgelegt wurde.

Der Befehl `OLD` löscht das Programm, das sich gerade im Arbeitsspeicher befindet, und liest das neue Programm ein. Anschliessend kann es mit `RUN` ausgeführt werden.

So kann das in den vorigen Abschnitten abgespeicherte Programm wieder geladen und ausgeführt werden durch:

[6] Über die Kennung `C` wird der Rechner *CLARA* angesprochen, der für alle HKR-Rechner gemeinsam die Dateien verwaltet.

```
>OLD BEISPIEL_XZ
>RUN
    21.0000
>
```

Um sich die Arbeit zu erleichtern, können Programme auch mit dem Befehl `RUN` eingelesen werden. In diesem Fall ist hinter dem Befehlswort `RUN` der Name des Programms anzugeben. Dadurch wird das Programm eingelesen (wobei wie bei dem Befehl `OLD` ein altes Programm, das sich noch im Arbeitsspeicher befindet, gelöscht wird) und sofort nach dem Einlesen ausgeführt.

Das Einlesen und Abarbeiten des Programms `BEISPIEL_XZ` könnte man also auch mit einem Befehl erreichen:

```
>RUN BEISPIEL_XZ
    21.0000
>
```

Arbeitet man gerade auf einem HKR-Rechner und möchte ein Programm ausführen, das auf dem Rechner *ALICE* abgespeichert wurde, ist, wie auf Seite 40 erwähnt, dem Namen die Kennung `A::` voranzustellen (und umgekehrt die Kennung `C::`, um von dem Rechner *ALICE* auf Programme auf den HKR-Rechnern zuzugreifen):

```
>RUN A::BEISPIEL_XZ
    21.0000
>
```

Als Ergänzung sollte noch erwähnt werden, dass es noch einen weiteren Befehl zum Einlesen von Programmen gibt, den Befehl `LOAD`. Im Gegensatz zum Befehl `OLD` wird hier ein Programm, das sich bereits im Arbeitsspeicher befindet, nicht zuvor gelöscht. Es werden lediglich neue Zeilen dazugefügt (und dabei Zeilen, die in beiden Programmen vorkommen, durch die neu eingelesenen ersetzt).

7.6.3 Löschen des NODAL-Arbeitsspeichers

Soll ein neues NODAL-Programm erstellt werden, ist zuvor das alte zu löschen. Das kann geschehen, indem alle Zeilennummern allein (ohne Befehl dahinter) eingegeben werden. Besonders bei längeren Programmen ist dieses Verfahren natürlich sehr umständlich. Daher gibt es einen eigenen NODAL-Befehl zum Löschen des Speichers. Er lautet `ERASE ALL` und löscht alle Programmzeilen sowie alle bereits definierten Variablen[7].

7.6.4 Ausführen von Allgemeinprogrammen

Nicht jeder Benutzer wird selber Programme in NODAL erstellen müssen oder wollen, denn bereits der Direkt-Modus ist recht mächtig. Darüberhinaus stehen für viele Probleme von allgemeinem Interesse bereits fertige NODAL-Programme zur Verfügung, die jeder benutzen kann. Nach dem bisher behandelten sollte deutlich geworden sein, wie diese Programme aufzurufen sind (und was dabei in NODAL passiert). Trotzdem wird es noch einmal kurz zusammengestellt.

Bei den Allgemeinprogrammen handelt es sich überwiegend um einfache Programme zur Bedienung von Geräten und um Diagnoseprogramme. Damit sie von jedem benutzt werden können, bekamen sie einen Namen und wurden (wie beim Befehl `SAVE` erläutert) unter diesem Namen abgespeichert. Sie können über diesen Namen in den NODAL-Arbeitsspeicher geholt und dann ausgeführt werden.

[7] Wie die Formulierung des Befehls mit zwei Worten vermuten lässt, gibt es für das Löschen verschiedene Varianten. So ist es unter anderem auch möglich, nur Variablen (einzelne oder alle) oder nur das Programm (jeweils teilweise oder vollständig) zu löschen. Für Einzelheiten sei auf das Handbuch [2] verwiesen

Am einfachsten geschieht das mit dem Befehl `RUN <Programmnamen>`, der ein Programm lädt und es sofort ausführt.

Als Beispiel sei ein Programm erwähnt, mit dem festgestellt werden kann, an welche Steuereinheit (SE) ein Gerät angeschlossen ist. Von dem Gerät ist ja nur der Name bekannt; wo es steht und wie es angeschlossen ist, braucht den Benutzer nicht zu interessieren. Bei Störungen kann es aber schon wichtig sein, wo dieses Gerät angeschlossen ist. Um das zu ermitteln, steht ein Programm zur Verfügung, dass zu jedem Gerät (anzugeben über die Nomenklatur) die Nomenklaturen des Gruppenmikros und der Steuereinheit liefert, über die es angeschlossen ist. Dieses Programm heißt `GETSE` und kann gestartet werden durch `RUN GETSE`. Als Beispiel sei der Bumpmagnet `S03MB4` gewählt:

```
>RUN GETSE
Please enter devicename (<CR> to end)  : S03MB4 <RETURN>
Nomen of GuP: S00CG63_
Nomen of SE : S00CS632
Please enter devicename (<CR> to end)  : <RETURN>
>
```

Hieran erkennt man, dass er über den Gruppenmikro S00CG63_ und die SE S00CS632 angeschlossen ist.

7.7 Editor

Zur Eingabe von Programmen wurde bisher nur erwähnt, dass man sie direkt eintippen kann. Änderungen von bereits bestehenden Zeilen können durchgeführt werden, indem man sie einfach neu eingibt. Dieses Verfahren ist natürlich sehr mühsam, und es gibt für diese Aufgabe etwas Besseres, nämlich Editoren. Davon sind in NODAL zwei vorgesehen: Ein spezieller, der nur in NODAL verfügbar ist, und der „Language Sensitive Editor“, abgekürzt LSEDIT, der der Standard-Editor auf der VAX ist.

An dieser Stelle soll nur der VAX-Editor LSEDIT erwähnt werden. Mit ihm kann das gerade im NODAL-Arbeitsspeicher befindliche Programm abgeändert werden oder auch ein neues erstellt werden, wenn der Arbeitsspeicher noch kein Programm enthält.

Für alle, die diesen Editor noch nicht kennen, soll an dieser Stelle eine wirklich sehr kurze Einführung gegeben werden. Um mit diesem Editor bereits arbeiten zu können, muss man nur 7 Tasten kennen — neben denen der Schreibmaschinentastatur zum Eingeben des Textes. Diese Tasten sollen nun erläutert weden.

Wichtig ist, dass die Tasten im Ziffernblock rechts neben der eigenlichen Tastatur im LSEDIT *nicht* die Ziffern darstellen, sondern Kommandos bedeuten. Will man Ziffern eingeben, sind die Tasten der eigentlichen Schreibmaschinen-Tastatur zu benutzen.

7.7.1 Starten des LSEDIT

Zum Starten dieses Editors dient der Befehl `LSEDIT`. Dieser Befehl kann nicht abgekürzt werden.

Nach dem Starten des Editors wird das gerade im NODAL-Arbeitsspeicher befindliche Programm auf dem Bildschirm angezeigt, so ähnlich, wie es bei dem NODAL-Befehl `LIST` erscheint. Das Starten des LSEDIT kann etwas dauern, also nicht ungeduldig werden.

7.7.2 Beenden des LSEDIT

Der wichtigste Befehl des LSEDIT ist zunächst der, mit dem man ihn wieder verlassen kann.

Er sieht etwas umständlich aus, aber auf die gezeigte Weise sollte er immer funktionieren. Zur Arbeitsersparnis gibt es auch noch einen kürzeren, der im Anschluss erwähnt werden soll.

Es ist *nacheinander* folgendes zu tun:

1. im Ziffernblock (rechts) die Taste `PF1` drücken,
2. im Ziffernblock (rechts, *nicht* in der Schreibmaschinentastatur) die Taste `7` drücken,
3. den Text `EXIT` eintippen (erscheint in der Kommando-Zeile, das ist die drittletzte Zeile auf dem Bildschirm),
4. den Text durch die Taste `ENTER` (rechts unten im Ziffernblock) abschliessen. Hier ist meist auch die Taste `RETURN` möglich.

Mit dieser Sequenz wird der Editor verlassen und das Programm aus dem Editor in den NODAL-Arbeitsspeicher übertragen.

In den meisten Fällen (aber unter Umständen funktioniert es nicht immer!) kann man den Editor genauso durch folgende Sequenz verlassen:

1. im Ziffernblock (rechts) die Taste `PF1` drücken,
2. in der Schreibmaschinentastatur die Taste `X` (für „exit“) drücken.

Das waren schon 3 der besonderen Tasten (und noch eine nützliche zur Arbeitsersparnis).

7.7.3 Eingeben und Ändern

Im LSEDIT sieht man, wie auch in NODAL, eine Schreibmarke. Texte können jeweils an der Stelle eingefügt oder gelöscht werden, an der die Schreibmarke steht. Will man etwas ändern oder neu eingeben, muss erst die Schreibmarke an die entsprechende Stelle gefahren werden.

Wie in NODAL kann man die Schreibmarke mit den Richtungstasten bewegen. Während man aber dort die Schreibmarke nur innerhalb der Eingabezeile bewegen kann, kann man die Schreibmarke im LSEDIT zusätzlich auch vertikal verfahren, wie auf den Tasten angegeben, und sich damit beliebig im Programm bewegen.

Jeweils an der Stelle der Schreibmarke kann neuer Text eingefügt werden. Zum Beenden einer Zeile und Einfügen einer neuen dient die Taste `RETURN`.

Natürlich muss man auch einmal etwas Löschen. Dazu dient die Löschtaste rechts oben in der Schreibmaschinentastatur, die etwa wie `<X|` beschriftet ist. Mit ihr wird das Zeichen links von der Schreibmarke gelöscht. Steht die Schreibmarke am Beginn einer Zeile, bewirkt das Drücken der Taste `RETURN` das Aufheben der Zeilentrennung.

Damit wären die allerwichtigsten Sondertasten des LSEDIT erläutert.

7.7.4 Weitere Befehle

Der LSEDIT kennt sehr sehr viele Befehle, die zum Teil sehr mächtig sind und mit denen man sich die Arbeit sehr erleichtern kann. Die wichtigsten davon können durch Drücken von Sondertasten (z.B. die Tasten im Ziffernblock) aufgerufen werden, so dass nicht viel Tipparbeit erforderlich ist.

Wer mit dem LSEDIT arbeiten möchte, sollte sich über die Belegung dieser Sondertasten informieren. Wenn man den LSEDIT aufgerufen hat (also nicht im eigenlichen NODAL), kann man einen ersten Überblick durch Drücken der Taste `PF2` im Ziffernblock erhalten. Es wird dadurch in Kurzform die Belegung der Sondertasten angezeigt.

Dabei wird man bei den meisten Tasten zwei Befehle finden. Den ersten Befehl erhält man, wenn man die entsprechende Taste direkt drückt. Um den zweiten Befehl zu erhalten, der invers dargestellt ist, muss zuvor die als `GOLD` bezeichnete Taste `PF1` gedrückt werden.

Das wurde schon ausgenutzt, um den Editor zu verlassen. Dazu wurde das LSEDIT-Kommando `EXIT` verwendet. Um das eingeben zu können, mussten die Tasten `GOLD` (`PF1`) und `7` gedrückt werden.

Wie man mit den Erläuterungen in der durch `PF2` aufgerufenen Kurzerklärung umgeht, ist jeweils knapp beschrieben. Falls man es nicht gleich findet: Verlassen kann man diese Kurzerklärung durch Drücken der Taste `RETURN`.

8 Weitere Befehle und Erläuterungen

Nachdem die Grundprinzipien der Programmierung in NODAL erläutert wurden, sollen nun die wichtigsten der noch nicht behandelten NODAL-Befehle vorgestellt werden. Wie bereits erwähnt, können sie sowohl in Programmen als auch interaktiv im Direkt-Modus benutzt werden. Ferner soll vertiefend auf die Bedeutung der Zeilennummern eingegangen werden.

8.1 Kommentare

Oft ist es sehr hilfreich, Programme durch zusätzliche Texte näher zu erläutern. Das gilt insbesondere für NODAL, da hier die Programme nicht besonders gut lesbar sind.

Um erläuternden Text einfügen zu können, muss dieser besonders gekennzeichnet sein. Dazu dient das Prozent-Zeichen (`%`), das statt eines Befehlswortes an erster Stelle stehen muss. Alles, was danach in der Zeile (bis zum Zeilenende) folgt, wird als Kommentar aufgefasst und von NODAL nicht weiter interpretiert.

Ein Ausschnitt aus einem Programm könnte also etwa so aussehen:

```
>37.25 % nun werden die Stromstaerken wichtiger Magnete angezeigt
>37.30 TYPE TK1MU2(CURRENTI)
>37.35 TYPE TK7MU5(CURRENTI)
```

Es ist auch möglich, hinter einen Befehl noch einen Kommentar zu schreiben. Dazu dient die Folge von Semikolon und Prozentzeichen (`;%`):

```
>TYPE TK1MU2(CURRENTI);% dieser Umlenker ist wichtig!
```

Alles, was nach den Zeichen „`;%`“ bis zum Zeilenende folgt, wird nicht weiter von NODAL interpretiert.

8.2 Befehl `ASK`

Bisher wurde gesagt, dass mit dem Befehl `SET` Variablen numerische Werte zugewiesen werden können. Das bedeutet aber in Programmen, dass bei jedem Abarbeiten derselbe Wert verwendet wird. Oft ist es dagegen erforderlich, dass der Wert vom Benutzer bei jedem Abarbeiten verschieden gesetzt wird. Dazu muss der Wert vom Programm abgefragt werden können, wozu in NODAL der Befehl `ASK`vorgesehen ist.

Hierbei ist hinter dem Befehlswort `ASK` der Name der Variablen anzugeben, der der Wert zugewiesen werden soll. Zusätzlich kann erläuternder Text angegeben werden, der bei jedem Abarbeiten des Befehls mit ausgegeben wird. Dieser Text ist in Hochkommas einzuschliessen, ähnlich wie beim Befehl `TYPE`. Mit diesem Text kann man etwa kennzeichnen, um welche Variable es sich handelt oder an welcher Stelle vom Programm man sich gerade befindet. Das ist natürlich besonders hilfreich, wenn in einem Programm mehrere Abfragen enthalten sind.

Der Befehl `ASK` kann abgekürzt werden bis auf `A`.

So könnte man sich ein kleines Programm schreiben, um einen Magneten zu optimieren:

```
>15.20 ASK "neuer Wert fuer Steerer" NewValue
>15.30 SET TK5MS6H(CURRENTS)=NewValue
>RUN
neuer Wert fuer Steerer : 17
>RUN
neuer Wert fuer Steerer : 37
>
```

Bei jedem Abarbeiten des Programms wird ein Wert für den Strom des Steerers abgefragt und der eingegebene Wert gesetzt.

8.3 Befehl `IF`

Bisher wurde gesagt, dass Programme Stück für Stück von vorne bis hinten abgearbeitet werden. Das heißt aber, dass immer alle Befehle des Programms abgearbeitet werden. Oft möchte man aber Befehle nicht immer ausführen lassen, sondern nur in manchen Fällen.

In NODAL ist dazu ein Vergleichsbefehl vorgesehen, mit dem Befehle abhängig von einer Bedingung ausgeführt werden. Er ist durch das Befehlswort `IF` beschrieben, auf das ein Vergleich folgen muss. Danach folgt, durch ein Semikolon abgetrennt, ein NODAL-Befehl.

In einer formalen Beschreibung ist der Befehl gegeben durch

```
IF <Vergleich> ; <Befehl>
```

Der Vergleich besteht aus zwei numerischen Ausdrücken, getrennt durch einen der Vergleichsoperatoren

- `>` (größer),
- `<` (kleiner),
- `>=` (größer oder gleich),
- `<=` (kleiner oder gleich),
- `=` (gleich) und
- `<>` (ungleich).

Ist der Vergleich wahr, wird der hinter der Abfrage folgende Befehl ausgeführt. Dieser Befehl muss in der gleichen Zeile wie die Abfrage stehen! Der Befehl `IF` kann nicht abgekürzt werden.

Zur Illustration sei die folgende Sequenz angegeben, die im Direkt-Modus eingegeben werden kann:

```
>SET TestWert=17
>IF TestWert>=17;TYPE "Bedingung ist wahr"
Bedingung ist wahr
>IF TestWert<17;TYPE "Bedingung ist wahr"
>
```

Beim ersten Mal ist die Bedingung in der Abfrage wahr und der folgende `TYPE`-Befehl wird ausgeführt, also der Text ausgegeben. Bei der zweiten Abfrage ist die Bedingung falsch und der `TYPE`-Befehl wird nicht ausgeführt.

Das genannte Beispiel ist natürlich nicht sehr sinnvoll, da man schon vorher weiß, was passieren wird. Ein realistischeres Beispiel wäre etwa, in einem Programm einen Gerätestatus abzufragen. Dieser Status ist in Ordnung, wenn alle Bits des Statusworts auf Eins gesetzt sind. Der Status ist ein

32-Bit Wort. Sind alle Bits gesetzt, hat es in hexadezimaler Darstellung den Wert $\texttt{0FFFFFFFF}_{\texttt{hex}}$, der in NODAL mit der Formatierung `[[` für hexadezimale Zahldarstellung angegeben werden kann als `[[FFFFFFFF`. Eine Abweichung von diesem Wert bedeutet einen Fehler, worauf in einem Programmteil hingewiesen werden könnte durch:

```
  ⋮
17.50 IF TK5MS6V(STATUS)<>[[FFFFFFFF;TYPE "Fehler im Steerer TK5MS6V" !
  ⋮
```

Hier wird das Statuswort des Magneten gelesen und eine Meldung angezeigt, wenn nicht alle Bits gesetzt sind, also der Wert ungleich $\texttt{0FFFFFFFF}_{\texttt{hex}}$ist.

Nach der Abfrage nur einen Befehl ausführen zu können, ist meistens zu wenig. Wie schon beim Befehl `FOR` erwähnt wurde, können entsprechend auch auf die `IF`-Abfrage nicht nur ein Befehl, sondern auch mehrere folgen. Welche Möglichkeiten dazu in NODAL bestehen, wird bald gezeigt werden.

8.4 Befehl `END`

Beim Abarbeiten eines NODAL-Programms werden alle Zeilen, angefangen mit der kleinsten Zeilennummer, nacheinander abgearbeitet bis die letzte erreicht ist.

Über den Befehl `END` (abkürzbar bis auf `EN`) kann man die Ausführung eines Programms auch schon vorzeitig beenden. Mit diesem Befehl wird das Programm abgebrochen, die noch folgenden Befehle werden nicht mehr ausgeführt.

Das erscheint auf den ersten Blick nicht sehr sinnreich — schließlich brauchen Befehle, die nie ausgeführt werden sollen, gar nicht erst hingeschrieben zu werden. Aber in Zusammenhang mit dem Befehl `IF` besteht nun die Möglichkeit, ein Programm unter bestimmten Bedingungen vorzeitig abzubrechen. So ist etwa denkbar, bei einem Gerätezugriff zuerst den Gerätestatus zu lesen und den eigentlichen Zugriff nur durchzuführen, wenn dieser Status anzeigt, dass mit dem Gerät alles in Ordnung ist.

Das ließe sich durch folgendes kleines Programm realisieren:

```
1.10 TYPE "neuen Wert fuer Steerer setzen" !
1.20 IF TK5MS6H(STATUS)<>[[FFFFFFFF;END
1.30 TYPE "Steerer ist OK" !
1.40 ASK "neuer Wert" StromNeu
1.50 SET TK5MS6H(CURRENTS)=StromNeu
```

Die Zeilen `1.30` bis `1.50` werden nur ausgeführt, wenn alle Bits im Statuswort auf Eins gesetzt sind, was anzeigt, dass der Magnet einwandfrei arbeitet. Ist das nicht der Fall, beendet `END` den Programmablauf.

8.5 Befehl `DO`

Wie bereits erwähnt, werden eigentlich alle Befehle eines NODAL-Programms nacheinander ausgeführt. Mit den Befehlen `IF` und `END` gibt es zwar schon einige Möglichkeiten, diese Reihenfolge zu beeinflussen, aber sie sind doch nur sehr eingeschränkt. Darüberhinaus gibt es in NODAL viel mächtigere Befehle, den Programmablauf zu verändern.

Einer dieser Befehle ist `DO` (nicht abkürzbar). Mit ihm kann man (unter anderem, weiteres später) einzelne Zeilen gezielt ausführen. Dazu ist die Nummer der Zeile, die ausgeführt werden soll, hinter

dem Befehlswort anzugeben. Mit dem Befehl `DO` wird die dort angegebene Zeile ausgeführt und die Programmabarbeitung anschließend mit der auf `DO` folgenden Zeile fortgeführt.

Zur Illustration sei das folgende kleine Programmbeispiel gegeben:

```
>1.10 TYPE "Das ist" !
>1.20 DO 1.50
>1.30 TYPE "Reihenfolge" !
>1.40 END
>1.50 TYPE "die richtige " !
>RUN
Das ist
die richtige
Reihenfolge
>
```

Der Befehl `DO 1.50` in Zeile `1.20` bewirkt, dass die Zeile `1.50` vor der Zeile `1.30` ausgeführt wird.

Der Befehl `END` in Zeile `1.40` ist wichtig: würde er weggelassen, würde die Zeile `1.50` zum Schluss *noch einmal* ausgeführt (wie es der normalen Reihenfolge entspricht).

8.6 Befehl `GOTO`

Auf diesen Befehl soll nur am Rande hingewiesen werden, da man mit ihm Programme sehr unübersichtlich gestalten kann und er deshalb in modernen Programmiersprachen schon gar nicht mehr vorgesehen ist. Da NODAL aber nicht viele Möglichkeiten der Strukturierung besitzt, ist dieser Befehl manchmal ganz hilfreich.

Er bewirkt, dass die Programmausführung nicht mit dem nächsten Befehl fortgesetzt wird, sondern mit einer beliebigen Zeile, die hinter dem Befehlswort `GOTO` anzugeben ist. Der Befehl ist abkürzbar bis auf `G`.

Als Beispiel sei ein kleines Programm angegeben, das eigenlich nur eine Laufanweisung darstellt, bei der lediglich die Laufvariable ausgegeben wird und so die Zahlen von 1 bis 5 auf dem Bildschirm ausgegeben weden.

```
>10.10 SET Var=1
>10.20 TYPE Var !
>10.30 SET Var=Var+1
>10.40 IF Var>5;END
>10.50 GOTO 10.20
>
```

Es wird eine Variable `Var` auf einen Wert von 1 gesetzt. Ab der Zeile `10.20` wird das Programm mehrfach durchlaufen: Es wird `Var` um 1 erhöht und in der Zeile `10.30` ausgegeben. Ist `Var` größer als 5, wird das Programm beendet (Zeile `10.40`). Die letzte Zeile bewirkt, dass die Programmausführung mit der Zeile `10.20` fortgesetzt wird, das Programm also ab dieser Zeile erneut durchlaufen wird (bis es nach dem fünften Durchlauf in der Zeile `10.40` beendet wird).

Achtung:

Durch hinreichend viele geeignet plazierte Sprungbefehle kann man jedes Programm beliebig unübersichtlich machen (Spagettikode). Daher ist der Befehl `GOTO` vorsichtig zu verwenden!

8.7 Zeilengruppen

Mit dem Befehl `DO` einzelne Zeilen auszuführen stellt nur eine kleine Ergänzung des sonst rein sequentiellen Programmablaufs dar. Aber dieser Befehl erlaubt auch, mehrere Befehle mit einem Aufruf abzuarbeiten.

Dabei soll der NODAL-Begriff der Zeilengruppe erläutert werden, mit dem die zunächst merkwürdig aussehende Form der Zeilennummer, bestehend aus Vor- und Nachkommaanteil, zusammenhängt. Diese Zeilengruppen sind eine Besonderheit von NODAL, die es etwa in BASIC nicht gibt.

8.7.1 Zeilennummern

Wie bereits erwähnt, ist durch die Zeilennummern die Reihenfolge bei der Programmausführung festgelegt. Anstatt dabei etwa vierstellige Nummern zuzulassen, die den gleichen Bereich überdecken würden, sind nur Zeilennummern mit zwei Vor- und zwei Nachkommastellen erlaubt.

Dadurch ist eine Untereinteilung der Programmzeilen möglich: Zeilen mit dem gleichen Vorkommaanteil bilden eine Zeilengruppe. Jede dieser Zeilengruppen ist gekennzeichnet durch den gemeinsamen Vorkommaanteil ihrer Zeilennummern, die Gruppennummer. Somit besteht jedes NODAL-Programm aus bis zu 99 Gruppen, von denen jede wiederum bis zu 99 Zeilen enthalten kann. Eine Zeile `17.04` liegt also in der Gruppe `17`.

Der erste Vorteil der Einführung von Gruppennummern besteht darin, dass die Übersichtlichkeit von Programmen etwas erhöht wird. Es wird vielleicht schon aufgefallen sein, dass mit dem Befehl `LIST` die einzelnen Programmgruppen durch je eine Leerzeile voneinander getrennt angezeigt werden.

Der größere Vorteil liegt aber darin, dass jede Gruppe als ganzes angesprochen werden kann, und zwar über ihre Gruppennummer. Diese Gruppennummer ist, wie bereits gesagt, eine ein- oder zweistellige Zahl ohne Nachkommaanteil.

8.7.2 Aufruf einer Zeilengruppe über `DO`

Mit dem Befehl `DO` einzelne Zeilen aufzurufen, schafft noch nicht viele Möglichkeiten. Die wesentliche Bedeutung dieses Befehls liegt darin, mit ihm eine ganze Zeilengruppe aufrufen zu können.

Dazu ist hinter `DO` statt der Zeilennummer (mit Nachkommaanteil) die gewünschte Gruppennummer (nur der Vorkommaanteil) anzugeben. Entsprechend einer einzelnen Zeile wird hierbei die ganze Zeilengruppe abgearbeitet und anschließend mit dem auf `DO` folgenden Befehl fortgefahren.

Auf diese Weise ist es möglich, Befehlsfolgen, die an mehreren Stellen im Programm benötigt werden, nur einmal (zusammengefasst als eine Zeilengruppe) aufzuschreiben und sie jeweils über `DO` aufzurufen. Die Zeilengruppen stellen somit eine einfache Form von Unterprogrammen dar, die etwa den Subroutines in BASIC entsprechen.

Es kann auch sinnvoll sein, nur einmal erforderliche Befehlsfolgen zusammenzufassen. Das ist wichtig, wenn in NODAL-Befehlen wie `FOR` und `IF` nur wenige Befehle angegeben werden können (wobei erst noch erklärt werden muss, wie mehr als ein Befehl anzugeben ist), aber eine längere Befehlskette erforderlich ist.

So lässt sich ein früher angegebenes Beispiel zur Einstellung eins Steerers, das für jeden neuen Wert erneut aufgerufen werden musste, damit etwas eleganter formulieren:

```
10.10 FOR Repetio=1,10000;DO 20
10.20 END

20.10 ASK "neuer Wert fuer Steerer" StromNeu
20.20 SET TK5MS6H(CURRENTS)=StromNeu
```

Die Gruppe 20 wird nun 10000 mal aufgerufen und jedesmal ein neuer Wert für den Steerer abgefragt sowie dieser gesetzt. Die Schleifengrenze von 10000 wurde nur deshalb so hoch gewählt, damit sie niemals erreicht wird (niemand wird 10000 mal versuchen, einen Magnetwert zu ändern). Damit man mit dem Programm auch wieder aufhören kann, steht ja die Flucht-Taste (`ESC`, `F11`) zur Verfügung, mit der man das Programm vorzeitig abbrechen kann.

Ein Aufrufen dieses Programms mit dreimaligem Setzen des Wertes und anschließendem Abbrechen durch die Flucht-Taste (`ESC`, `F11`) würde etwa folgende Ausgabe auf dem Bildschirm erzeugen:

```
>RUN
neuer Wert fuer Steerer : 17
neuer Wert fuer Steerer : 4
neuer Wert fuer Steerer : 37
neuer Wert fuer Steerer : <ESC>
***  NODAL ERROR  16   Escape Typed
>
```

Jede über `DO` aufgerufene Gruppe darf wiederum `DO`-Befehle enthalten. Insgesamt können etwa 45 `DO`-Befehle ineinander geschachtelt werden.

Und wie generell erlaubt, lässt sich `DO` auch im Direkt-Modus verwenden: damit ist es möglich, statt eines kompletten Programms (das sich gerade im Arbeitsspeicher befindet) nur eine einzelne Gruppe abzuarbeiten.

8.7.3 Befehl `RETURN`

Wird eine Zeilengruppe über den Befehl `DO` aufgerufen, wird sie normalerweise vollständig abgearbeitet. Erst nachdem die letzte Zeile dieser Gruppe durchlaufen wurde, wird mit dem auf den rufenden `DO`-Befehl folgenden Anweisung weitergemacht.

Man kann aber durch den Befehl `RETURN` (abkürzbar bis auf `RET`) erreichen, dass die Gruppe (oder auch die einzelne Zeile) vorzeitig verlassen wird. Er wirkt so, als ob die letzte Zeile erreicht worden ist.

Oft ist `RETURN` als letzter Befehl in einer Zeilengruppe aufgeführt. Das ist eigentlich nicht erforderlich, verdeutlicht aber, dass hier die Gruppe zu Ende ist.

8.7.4 Auflisten einer Gruppe

Auf Gruppen als Ganzes kann auch in vielen anderen Befehlen zugegriffen werden. Ein Beispiel dafür ist der Befehl `LIST`. Gibt man hinter dem Befehlswort noch eine Gruppennummer an, wird nur die entsprechende Gruppe angezeigt.

So lässt sich mit dem Beispielprogramm aus dem vorigen Abschnitt jeweils nur eine Gruppe auflisten wie hier die Gruppe 10:

```
>LIST 10
10.10 FOR Repetio=1,10000;DO 20
10.20 END
>
```

9 Zeichenketten

9.1 Was ist eine Zeichenkette

Bisher wurde in NODAL nur die Verarbeitung von numerischen Ausdrücken behandelt. Es ist darüberhinaus aber auch möglich, Texte zu verarbeiten. Erste Beispiele davon wurden schon in den Befehlen `TYPE` und `ASK` erwähnt, als gezeigt wurde, wie dort zusätzliche Texte ausgegeben werden.

Für solche Texte stehen in NODAL eigene Datentypen, die Zeichenketten (Englisch: „string“), zur Verfügung. Eine solche Zeichenkette ist eine Folge von fast beliebigen Zeichen der Tastatur, also Buchstaben, Ziffern und Sonderzeichen. Damit sie NODAL als solche erkennen kann, ist sie in einfachen (`'`) oder in doppelten (`"`) Hochkommas einzuschließen, wie es schon bei dem Befehl `TYPE` angegeben wurde.

Bei der Kennzeichnung von Zeichenketten können sowohl einfache als auch doppelte Hochkommas verwendet werden. Dabei ist eine Zeichenkette aber mit dem Zeichen abzuschließen, mit dem sie begonnen wurde, das jeweils andere darf in der Zeichenkette enthalten sein. So sind

`"NODAL Handbuch"` und `'NODAL Handbuch'`

sowie `"Sophokles' Werke"` oder `'Sie sagte: "Tolles NODAL!"'`

gleichermaßen erlaubt, nicht jedoch

`"NODAL Handbuch'` oder `'NODAL Handbuch"`

Maximal dürfen in einer Zeichenkette 80 Zeichen enthalten sein.

Zur Verarbeitung von Zeichenketten stehen in NODAL eigene Befehle zur Verfügung, von denen die wichtigsten zur Kennzeichnung mit einem Dollarzeichen beginnen (`$`).

Statt der Begriffe „Zeichenkette“ oder „string“ wird in den Beschreibungen über NODAL meistens der Begriff „Verkettung“ („concatenation“) verwendet, die eine Erweiterung der Zeichenketten darstellt. Beide Begriffe können weitgehend synonym benutzt werden, auf die Besonderheiten der Verkettungen wird noch eingegangen werden.

9.2 Befehl `$SET`

Wie erläutert wurde, kann man in NODAL Variable definieren und ihnen numerische Werte zuordnen. Entsprechendes ist auch mit Zeichenketten möglich, wobei der Mechanismus ähnlich ist wie bei numerischen Variablen.

Zeichenkettenvariable werden wie numerische Variable definiert, indem ihnen ein Wert zugewiesen wird. Der Befehl dafür lautet `$SET`, wobei das Dollarzeichen auf die Zeichenketten hinweist. Der Befehl ist abkürzbar bis auf `$S`.

So wird durch

```
>$SET Name="Albert Einstein"
>
```

eine Zeichenkettenvariable mit dem Namen `Name` angelegt, in der der Text `Albert Einstein` abgespeichert wurde. Durch eine erneute Zuweisung kann ihr eine andere Zeichenkette zugewiesen werden. Dabei ist zu beachten, dass einer bereits angelegten *numerischen* Variablen keine Zeichenkette zugewiesen werden kann, die Variable muss als Zeichenkettenvariable angelegt worden sein.

Um den Inhalt der Variablen anzusehen, kann der Befehl `TYPE` verwendet werden. Mit diesem Befehl können beliebig gemischt numerische Werte und Zeichenketten ausgegeben werden:

```
>$SET Text1="Dies "
>$SET Text2="  und"
>SET Vier=4
>TYPE  Text1  "ist"  17  Text2  4
Dies ist    17.0000  und     4.0000
>
```

9.3 Befehl `$ASK`

Analog zu dem Befehl `ASK` zur Eingabe von numerischen Werten steht ein entsprechender Befehl zur Eingabe von Zeichenketten zur Verfügung, der `$ASK` lautet. Der Befehl ist abkürzbar bis auf `$A`.

Er ist genauso zu benutzen wie der Befehl `ASK` für numerische Werte:

```
>$ASK "Namen angeben" Name
Namen angeben : Mickey Mouse
>TYPE Name
Mickey Mouse
>
```

9.4 Befehl `$IF`

Mit dem Befehl `IF` besteht die Möglichkeit, Anweisungen nur unter bestimmten Bedingungen auszuführen. Dazu wird ein Vergleich von numerischen Werten durchgeführt.

Mit dem Befehl `$IF` steht ein entsprechender Befehl zur Verfügung, bei dem der Vergleich zwischen zwei Zeichenketten (wobei auch Zeichenkettenvariablen erlaubt sind) durchgeführt wird.

Um ein Programm vom Bediener her zu beenden, könnte man etwa folgendes programmieren:

```
37.15 $ASK "Programm Beenden: J eingeben" Beenden
37.20 $IF Beenden='J';END
```

Beim Abarbeiten des Programms wird der Benutzer gefragt, ob er das Programm beenden möchte. Gibt er dann `J` für „Ja“ ein, wird es beendet, bei allen anderen Eingaben nicht[8].

9.5 Befehl `DIMENS-S`

Bisher wurden nur Felder von numerischen Werten behandelt. Es ist aber auch möglich, Felder mit Zeichenkettenelementen zu vereinbaren. Dazu dient der Befehl `DIMENS-S`. Dabei kann das Wort `DIMENS` bis auf `DI` abgekürzt werden, also sind auch erlaubt `DIMEN-S`, `DIME-S`, `DIM-S` und `DI-S`.

Hinter diesem Befehlswort ist der Name des Feldes anzugeben. Im Gegensatz zu Feldern numerischer Werte ist keine Anzahl von Elementen anzugeben. Bei Feldern von Zeichenketten können beliebige Elemente angesprochen werden. Wie bei den numerischen Feldern erfolgt der Zugriff auf die einzelnen Elemente, indem man hinter den Namen des Feldes in Klammern die Nummer des

[8] Hier muss das „J“ wirklich als Großbuchstabe eingegeben werden, damit der Vergleich wahr ist. Sollen auch Kleinbuchstaben erlaubt sein, müssen diese explizit abgefragt werden oder die Eingabe muss zuvor mittels der Funktion `CAP` (siehe [2]) in Großbuchstaben umgewandelt werden.

Elementes setzt. Ein Feldelement, dem noch kein Wert zugewiesen wurde, ist leer, d.h. es enthält keine Zeichen.

```
>DIMENS-S Dichter
>$SET Dichter(1)='Goethe'
>$SET Dichter(2)='Schiller'
>$SET Dichter(3)='Lessing'
>TYPE Dichter(1) '  ' Dichter(3)
Goethe  Lessing
>
```

9.6 Verkettung

Statt der Zeichenketten können in NODAL allgemein sogenannte Verkettungen (Englisch: „concatenation“) verwendet werden, die eine Erweiterung der Zeichenketten darstellen. Sie sind eine Eigenheit von NODAL, die es in anderen Programmiersprachen in dieser Form nicht gibt. Sie bedeuten im Wesentlichen, dass man Zeichenketten aneinanderfügen kann, indem man sie einfach hintereinander schreibt.

Statt

```
>$SET Name='Friedrich Schiller'
```

kann man auch schreiben:

```
>$SET Name='Friedrich'   ' '   'Schiller'
```

Beide Male enthält die Variable `Name` die gleiche Zeichenkette.

Auch Variableninhalte lassen sich verknüpfen:

```
>$SET Vorname='Friedrich'
>$SET Nachname='Schiller'
>$SET Name=Vorname   ' '   Nachname
>TYPE Name
Friedrich Schiller
>
```

Die Besonderheit der Verkettung besteht nun darin, dass Zeichenketten auch numerische Werte zugewiesen werden können (mit denen man dann allerdings nicht rechnen kann). Sie werden dabei zuvor in Zeichenketten umgewandelt, wobei die gleichen Formatierungsanweisungen benutzt werden können wie im `TYPE`-Befehl. Die numerischen Werte können dabei direkt als Ziffern angegeben werden, es können aber auch numerische Variablen benutzt werden. Als Beispiel sei einer Variablen `Spiel` nacheinander der numerische Wert 17 und der Inhalt einer Variablen zugeordnet:

```
>$SET Spiel=17
>TYPE Spiel
     17.0000
>SET Zahl=4
>$SET Spiel=Spiel   ' und'   Zahl
>TYPE Spiel
     17.0000  und     4.0000
>
```

Bei der Zuweisung eines numerischen Wertes wird durch die Unterscheidung zwischen dem numerischen Befehl `SET` und dem Zeichenkettenbefehl `$SET` festgelegt, um welchen Typ von Variablen es sich handelt.

Zur Demonstration der Formatierung sei die Umwandlung einer Zahl in hexadezimale Schreibweise erwähnt:

```
>SET Zahl=21
>$SET Ergebnis=Zahl   ' ist hexadezimal gleich '   ]]Zahl
>TYPE Ergebnis
     21.0000 ist hexadezimal gleich 00000015
>
```

Das Umgekehrte, nämlich eine als Zeichenkette dargestellte Zahl einer numerischen Variablen zuzuweisen, ist nicht direkt möglich. Hierbei muss die Umwandlung explizit über die in NODAL verfügbare Funktion `EVAL` (siehe [2]) durchgeführt werden.

10 Ergänzungen

10.1 Mehrere Befehle in einer Zeile

Es ist in NODAL sowohl im Direkt-Modus als auch in Programmen möglich, mehrere Befehle in eine Zeile zu schreiben. Dazu sind die einzelnen Befehle durch Semikola voneinander zu trennen. Statt

```
>SET Drei=3
>SET Wert=2*Drei
>TYPE Wert
      6.0000
>
```

kann man also auch schreiben:

```
>SET Drei=3;SET Wert=2*Drei;TYPE Wert
      6.0000
>
```

Wie hoffentlich schon an diesem Beispiel erkannt wird, sind Zeilen mit mehreren Befehlen meist schlecht zu überblicken. Daher sollte man sparsam mit der Verwendung mehrerer Befehle in einer Zeile sein. Oft ist es aber ganz hilfreich und erspart ihrerseits unübersichtliche Aufrufe über den Befehl `DO` oder sogar Sprünge, wenn etwa in einer Abfrage über `IF` nicht nur ein Befehl ausgeführt werden soll, sondern mehrere. Ein typischer Fall wäre etwa, bei einem Status, der einen Fehler anzeigt, nicht nur das Programm abzubrechen, sondern noch eine Meldung auszugeben:

```
17.50 IF TK5MS6V(STATUS)<>[[FFFFFFFF;TYPE 'Fehler in TK5MS6V';END
```

10.2 Befehl `WHILE`

Mit dem Befehl `FOR` wurde schon eine Möglichkeit gezeigt, Anweisungen wiederholt auszuführen. In NODAL steht dazu ein weiterer Befehl, der `WHILE`-Befehl (abkürzbar bis auf `WH`), zur Verfügung.

Mit ihm lässt sich ein Befehl so lange ausführen, wie ein Vergleich wahr ist. Formal sieht dieser Befehl so aus:

```
WHILE <Vergleich> ; <Befehl>
```

wobei im Vergleich zwei numerischer Ausdrücke mit den gleichen Operatoren wie im Befehl `IF`, also `>`, `<`, `>=`, `<=`, `<>` und `=`, verglichen werden. Der nach dem Semikolon folgende Befehl (oder mehrere, die dann durch Semikola getrennt werden müssen), wird so lange ausgeführt, wie der Vergleich wahr ist.

Als Beispiel seien alle Quadratzahlen unter 50 ausgegeben, und zwar im Direkt-Modus, um zu zeigen, wie einfach das geht. Es wurde dabei ausgenutzt, dass in dem `WHILE`-Befehl auch mehr als ein Befehl ausgeführt werden kann.

```
>SET Zahl=1
WHILE Zahl*Zahl<50;TYPE Zahl*Zahl !;SET Zahl=Zahl+1
     1.0000
     4.0000
     9.0000
    25.0000
    36.0000
    49.0000
>
```

Die Variable `Zahl` ist lediglich zu Beginn mit einem Wert zu versehen. In dem eigentlichen `WHILE`-Befehl wird ihr Quadrat ausgegeben (mit der Formatierung `!` für einen Zeilenvorschub) und sie anschließend um eins erhöht, damit beim nächsten Durchlauf die nächste Quadratzahl ausgegeben wird. Diese beiden Befehle werden so lange wiederholt, wie das Quadrat von `Zahl` kleiner ist als 50 (`WHILE Zahl*Zahl<50`).

10.3 Befehl `WAIT-T`

Oft besteht der Wunsch, sich periodisch etwas anzeigen zu lassen. Dazu ist es ganz hilfreich, die Möglichkeit zu haben, zwischen zwei Ausgaben etwas zu warten. Wichtig ist das insbesondere, um die Rechner nicht durch ständig ablaufende Programme unnötig zu belasten.

In NODAL steht dafür der Befehl `WAIT-T` zur Verfügung, wobei das Wort `WAIT` abgekürzt werden kann bis auf `WA` , also insgesamt `WA-T`. Hinter dem Befehl ist eine Zeitdauer in Sekunden anzugeben. Bei diesem Befehl wartet NODAL die angegebene Zeitdauer und fährt erst danach mit dem nächsten Befehl fort (oder ist, im Direkt-Modus, erst danach wieder bereit für Eingaben).

Eine Anwendung dieses Befehls ist etwa, periodisch bei einem Gerät nachzusehen, um zum Beispiel seinen Gerätestatus auszuwerten. Als Beispiel wird hier alle 10 Sekunden von dem Umlenker `TK2MU3` der Gerätestatus auf dem Bildschirm ausgegeben:

```
1.10 WHILE 1=1;TYPE ]]TK2MU3(STATUS) !;WAIT-T 10
```

Da die Bedingung in dem `WHILE`-Befehl (`1=1`) immer erfüllt ist, läuft dieser Befehl immer weiter. Er ist mit der Flucht-Taste (`ESC`, `F11`) abzubrechen[9].

10.4 Befehl `DIMENS-I`

NODAL arbeitet bei Rechnungen nur mit Gleitkommazahlen. So sind auch numerische Variablen generell von diesem Typ. Daneben besteht aber die Möglichkeit, Felder aus ganzzahligen Elementen anzulegen. Sie werden mit dem Befehl `DIMENS-I` vereinbart (abkürzbar bis auf `DI-I`).

Die einzelnen Feldelemente können genauso behandelt werden wie die Elemente der durch `DIMENS` angelegten Gleitkommafelder, wobei bei Wertzuweisungen gegebenenfalls gerundet wird.

Solche Ganzzahlfelder werden in vielen Fällen als Parameter von NODAL-Funtionen gefordert, insbesondere bei Funktionen des SIS-Kontrollsystems. Der Grund liegt darin, dass in früheren Versionen von NODAL nur mit einfacher Genauigkeit (etwa 7 Stellen) gerechnet wurde. Daher wurden die Ganzzahlfelder benutzt, um dennoch 32-Bit Werte ohne Rundungsfehler an das Kontrollsystem übertragen zu können. Damit nicht die bereits existierenden Programme umgestellt werden müssen, wurde dieser Parametertyp beibehalten.

[9] Achtung! Die Ausführung des Befehls `WAIT-T` kann nicht mit der Flucht-Taste (`ESC`, `F11`) unterbrochen werden. NODAL wartet stets die angegebene Zeit ab und unterbricht erst dann.

10.5 Befehl HELP

Wie bisher ersichtlich war, gibt es in NODAL sehr viele Befehle und Befehlsvarianten. Bei der großen Zahl fällt es schwer, sich alles zu merken — besonders, wenn nur gelegentlich mit NODAL gearbeitet wird. Um unklare Dinge nachsehen zu können, ist also eine Beschreibung erforderlich, die man aber nicht immer dabei hat.

Mit dem Befehl `HELP` (nicht abkürzbar) kann man in NODAL eine im Rechner gespeicherte Beschreibung aufrufen. Man erhält eine kurze Erläuterung und eine Liste von Auswahlpunkten. Durch Eingabe eines dieser Auswahlpunkte erhält man eine Beschreibung darüber und in der Regel weitere Unterpunkte, über die man sich genauso näher informieren kann. Durch Drücken der `RETURN`-Taste (ohne Angabe eines Auswahlpunktes) gelangt man jeweils eine Ebene höher in der Struktur und schließlich zurück nach NODAL.

Nach dem Aufruf der `HELP`-Funktion erscheint die folgende Erläuterung und Auswahlliste:

```
NODAL

  The NODAL language and interpreter has been designed to provide quick
  and easy access to all the components of the SIS/ESR Control System.
  To invoke NODAL use the following format:

  NODAL [qualifier[qualifier]]



  Additional information available:

  qualifiers
  /CAPITALIZE           /LONGCOMMAND
  Syntax     Operators  Commands   NODAL-Functions       File-IO-Functions
  Control-System-Functions         DBS-Functions         VMS-Functions
  Keyboard-Commands     Screen-Editor         General-Info
  Version-Guide         Bibliography          Buglist    Updates


NODAL Subtopic?
```

Erläuterungen über die einzelnen Unterpunkte erhält man durch Eingabe des entsprechenden Stichwortes (abschließen durch `RETURN`), also etwa durch `COMMANDS` über die NODAL-Kommandos. Dabei erscheinen in der Regel weitere Auswahlisten zur detaillierteren Information:

```
NODAL

  Commands

    At the present time the following commands are implemented
    in NODAL at GSI.



    Additional information available:
```

```
ASK       \$ASK      CALL      DEFINE    DIMENS    DO        \$DO
EDIT      END        ERASE     FOR       GOTO      IF        \$IF
LDEF      LIST       LOAD      \$MATCH    ?OFF      OLD       ?ON
OPEN      OVERLA     \$PATTE    QUIT      RETURN    ROF       RUN
SAVE      SDEF       SET       \$SET      TYPE      VALUE     \$VALUE
WAIT-T    WHILE      ZDEF
```

```
NODAL Commands Subtopic?
```

Über die einzelnen Kommandos könnte man sich nun entsprechend weiter informieren. Zurück zu NODAL gelangt man dann durch mehrfaches Drücken der `RETURN`-Taste ohne Eingabe eines Auswahlpunktes.

10.6 Formatierungen

Wie schon mehrfach erwähnt und teilweise auch schon benutzt, gibt es in NODAL viele Möglichkeiten, das Format von numerischen Werten bei Ausgaben zu verändern. Diese Formatierungen sollen an dieser Stelle weitgehend vollständig zusammengestellt und erläutert werden.

Dabei sei nicht nur an Ausgaben über den Befehl `TYPE` erinnert: die gleichen Formatierungen können sowohl bei Ausgaben über `TYPE` als auch bei Verkettungen benutzt werden[10].

Es gibt dabei verschiedene Formatierungsanweisungen:

Änderung der Zahlenbasis:
: Damit lassen sich numerische Werte binär, oktal oder hexadezimal ausgeben.

Änderung der Zahldarstellung:
: Es ist möglich, die Zahl der Stellen bei der Ausgabe zu verändern. So kann die Zahl der Nachkommastellen erhöht werden (oder ganz unterdrückt werden). Ferner können Werte explizit in exponentieller Darstellung ausgegeben weden.

Sonderzeichen:
: Gemeint ist hier ein Zeilenvorschub, das Einfügen von Leerzeichen und die Ausgabe von beliebigen Zeichen durch Angabe ihres ASCII-Wertes.

10.6.1 Änderung der Zahlenbasis

Normalerweise sind die Zahldarstellungen in NODAL dezimal. Insbesondere bei Gerätezugriffen hat man es aber oft mit Bitmustern zu tun — das Standardbeispiel ist das Statuswort. Solche Bitmuster werden durch Zahlen dargestellt. Um von solchen Zahlen wieder auf das Bitmuster zu schließen, kann man sie bitweise darstellen. Weiter sind auch hexadezimale und oktale Darstellungen möglich.

Um einen numerischen Ausdruck bitweise darzustellen, ist ein Fragezeichen (`?`) voranzustellen, zur oktalen Ausgabe eine schließende eckige Klammer (`]`) und zur hexadezimalen Ausgabe zwei (`]]`):

[10] Um bei dem Befehl `TYPE` Zahlenwerte auszugeben, wandelt NODAL intern die Werte in eine Verkettung um und benutzt dabei genau den gleichen Mechanismus wie im Befehl `$SET`.

```
>TYPE ?6
00000000000000000000000000000110
>TYPE ]10
00000000012
>TYPE ]]43
0000002B
>
```

Wie erwähnt, können statt Zahlen auch numerische Ausdrücke angegeben werden:

```
>SET Wert=5
>TYPE ]]6*Wert-2
0000001C
>TYPE ?6*Wert-2
00000000000000000000000000011100
>
```

Ausgegeben werden jeweils nur natürliche Zahlen, wobei zuvor bei Bedarf gerundet wird.

10.6.2 Änderung der Zahldarstellung

Die Zahldarstellung (Fließkommaformat oder Exponentialdarstellung, Zahl der dargestellten Stellen) wird durch Formatierungen verändert, die mit dem Prozentzeichen (`%`) beginnen.

Die beiden Formen dieser Formatierungen sind:

`%n`: eine ein- oder zweistellige Zahl nach dem Prozentzeichen.

`%n.m`: eine ein oder zweistellige Zahl nach dem Prozentzeichen, sowie durch einen Punkt abgetrennt eine zweistellige Zahl. Die Zahl nach dem Punkt ist *stets* zweistellig anzugeben, kleine Werte mit führender Null.

Mit diesen Formatierungen wird auch die Zahl der Zeichen angegeben, die der Wert bei der Ausgabe beansprucht. Ist die Zahl kürzer als die angegebene Ziffernzahl, wird sie rechtsbündig dargestellt. In binärer, oktaler und hexadezimaler Darstellung werden Nullen aufgefüllt, sonst Leerzeichen.

`%n`: ganzzahlige Darstellung
Hierbei wird nur der ganzzahlige Anteil (gerundet) eines Wertes dargestellt. Die Zahl der darzustellenden Ziffern einschließlich einem eventuellen Vorzeichen ist durch `n` gegeben. Diese Formatierung kann bei einer Änderung der Zahlenbasis zusätzlich angegeben werden, um bei kleinen Zahlen nicht so viele Nullen auszugeben.

```
>TYPE %3 2*7
 14
>TYPE %4 ]]29
001D
>
```

`%n.m`: Fließkommadarstellung
Hierbei werden Zahlen mit insgesamt `n` Zeichen (einschließlich einem eventuellen Vorzeichen und dem Dezimalpunkt) als Fließkommazahl dargestellt. Dabei werden `m` Nachkommastellen dargestellt. Zu beachten ist, dass die Nachkommastellen stets als *zweistellige* Zahl angegeben werden müssen! Zwei Nachkommastellen sind also als `02` anzugeben (in z.B. `6.2` wird die `2` als 20 Nachkommastellen aufgefasst)!

```
>TYPE %7.03 1/3-2
 -1.667
>$SET Verkett=%6.01 47.11
>TYPE Verkett
  47.1
>
```

`%0.m`: Exponentialdarstellung
Hierbei werden Zahlen in exponentieller Darstellung ausgegeben. Als Mantisse werden `m` Ziffern dargestellt. Wie bei der Fließkommadarstellung ist dieser Wert zweistellig anzugeben. Für das Vorzeichen, den Dezimalpunkt und den Exponenten werden insgesamt 6 Zeichen zusätzlich ausgegeben (eventuell als Leerzeichen). Die Angabe von Null für die Zahl der dargestellten Ziffern bewirkt die Ausgabe aller 16 signifikanten Ziffern.

```
>$SET Verkett=%0.04 2/3
>TYPE Verkett
 6.667E-1
>TYPE %0.0 PIE
 3.141592653589793E0
>
```

10.6.3 Sonderzeichen

Es ist möglich, beliebige ASCII-Zeichen in Zeichenketten einzufügen. Ferner gibt es besondere Formatierungen zum Einfügen von Leerzeichen und für einen Zeilenvorschub.

`\n`: ASCII-Zeichen
Es wird das ASCII-Zeichen eingefügt, dass durch die Zahl `n` (dezimal) kodiert ist. So ist das oft benötigte Zeichen `ESC` („escape“) durch 27 kodiert, ein „A“ hat den Wert 65, „B“ den Wert 66 usw.

```
>$SET Text= \78 \79 \68 \65 \76
>TYPE Text
NODAL
>
```

`&n`: Leerzeichen
Mit diesem Formatierungsbefehl kann man `n` Leerzeichen einfügen:

```
>TYPE 'Hier' &1 'ist' &2 'Platz' &3 'gelassen'
Hier ist  Platz   gelassen
>
```

`!`: neue Zeile
Dieser Formatierungsbefehl bewirkt, dass die Ausgabe auf der nächsten Zeile fortgesetzt wird.

```
>TYPE 'Jeweils' ! 'neue' !! 'Zeilen'
Jeweils
neue

Zeilen
>
```

10.7 Zahlbasisdeklaration

Alle Zahlen in NODAL werden normalerweise dezimal interpretiert. Es ist aber möglich, auch oktale oder hexadezimale Zahlen zu benutzen. Um sie als solche zu kennzeichnen, sind eine öff-

nende eckige Klammer ([) für oktale Darstellung und zwei öffnende eckige Klammern ([[) für hexadezimale Darstellung voranzustellen.

```
>TYPE %3 [10
  8
>TYPE %3 [[10
 16
>
```

Diese Darstellungen können auch in Ausdrücken benutzt werden:

```
>TYPE %3 2*[10
 16
>TYPE %3 5*[[10+2
 82
>
```

Im ersten Befehl wird $2 \cdot 10_{oct}$ berechnet, im zweiten $5 \cdot 10_{hex} + 2$.

Diese Zahlbasisdeklarationen sehen zwar formal sehr ähnlich aus wie Formatierungen im oktalen oder hexadezimalen Format, bedeuten aber etwas wesentlich Verschiedenens. Formatierungen sind nur im Befehl `TYPE` oder in Verkettungen möglich, Zahlbasisdeklarationen überall, wo Zahlen verwendet werden.

10.8 Unterprogramme

In NODAL stellen die Zeilengruppen bereits eine einfache Form von Unterprogrammen dar, die allerdings nicht die Anforderungen erfüllen, die eigentlich an Unterprogramme zu stellen sind. So ist keine Parameterübergabe möglich, und die Zeilengruppen sind nicht abgeschlossen (alle Variablen sind im gesamten Programm global, Befehle in Zeilengruppen können neben dem „Unterprogramm-“Aufruf durch `DO` auch auf andere Weise ausgeführt werden).

Neben den Zeilengruppen sind in NODAL auch Unterprogramme im eigentlichen Sinnen vorgesehen, die benutzerdefinierten Funktionen („Defined Functions“). Sie besitzen einen eigenen Datenbereich und der Austausch von Werten mit dem Hauptprogramm erfolgt über eine Parameterliste.

Da die Erläuterung dieser Funktionen den Rahmen dieser Schrift sprengen würde, sei diesbezüglich auf die ausführliche Dokumentation verwiesen ([2], [3], [4]).

11 Variable Gerätezugriffe

Bisher haben wir nur Formulierungen von Gerätezugriffen kennengelernt, in denen die Nomenklatur eines Gerätes immer explizit angegeben werden musste. Schreibt man sich kleine Programme in NODAL, so erreicht man bald den Punkt, wo es sehr hilfreich wäre, auch Nomenklaturen in Variablen ablegen und mit diesen Variablen einen Gerätezugriff ausführen zu können.

Diese Möglichkeit bietet NODAL. Wie das geht, wollen wir uns an einem kleinen Beispiel ansehen. Das Programm fragt nach der Nomenklatur eines Gerätes, ließt anschließend den Status dieses Gerätes und gibt ihn aus.

```
>1.10 $ASK 'Lese Status des Geraetes' nomen
>1.20 SET stat = $nomen(STATUS)
>1.30 TYPE 'Status von ' nomen ' ist: ' ?stat !!
>1.40 GOTO 1.10
```

Zunächst wird mit `$ASK` nach einer Zeichenkette, in unserem Fall nach der Nomenklatur eines Gerätes, gefragt. Die Benutzereingabe wird der Variablen `nomen` zugewiesen. Dies haben wir bereits kennengelernt.

Worauf man achten muss, ist die Formulierung `$nomen` im Gerätezugriff. Normalerweise steht an dieser Stelle des Datamodules die Nomenklatur eines Gerätes. In diesem Fall ein `$` direkt gefolgt von dem Variablennamen `nomen`. Diese Art der Formulierung, die man verallgemeinert mit

```
$<Variablenname>
```

beschreiben kann, besagt folgendes: Das `$` gibt an, dass direkt anschließend keine Nomenklatur sondern der Name einer Variablen folgt. Diese Variable muss eine Nomenklatur enthalten.

Die Variable in unserem Programm heißt `nomen`. Enthält `nomen` die Zeichenkette „`TK1MU1`“, so wird auf dieses Gerät, also auf `TK1MU1`, zugegriffen.

Würden wir das `$` vor `nomen` bei der Formulierung des Gerätezugriffs weglassen, so würde NODAL nicht auf `TK1MU1` zugreifen wollen, sondern eben auf das Gerät `NOMEN`, was es natürlich nicht gibt.

Lässt man das Programm laufen, so sieht das folgendermaßen aus:

```
>RUN
Lese Status des Geraetes : TK1MU1
Status von TK1MU1 ist: 11101001111101111101111111000111

Lese Status des Geraetes :
```

Diese Art der Formulierung, die Nomenklatur über eine Variable anzugeben, ist bei allen Gerätezugriffen möglich. Solche Formulierungen heißen in NODAL Indirections.

12 Fehlerbehandlung in Programmen

Auftretende Fehler führen in NODAL normalerweise zur Ausgabe einer Fehlermeldung und zur Rückkehr in den Direkt-Modus. Hat man ein Programm geschrieben und es tritt ein Fehler während der Abarbeitung des Programmes auf, wird das Programm unterbrochen und NODAL kehrt aus dem Programm-Modus zurück in den Direkt-Modus.

Das kann sehr hinderlich sein, wenn man in einem Programmablauf bewusst Fehler in Kauf nehmen will, oder wenn man auf diese Fehler in einer bestimmten Art und Weise reagieren möchte. Daher stellt NODAL für die Behandlung von Fehlern, die während Programmabläufen auftreten, einige Möglichkeiten zur Verfügung.

12.1 Fehlerbehandlung bei Gerätezugriffen

Angenommen, wir möchten den Status eines Gerätes jede Minute überprüfen. Dazu schreiben wir uns ein kleines Programm, das in einer Endlosschleife[11] alle 60 Sekunden einen Gerätezugriff macht, um den Status zu lesen und auszugeben.

```
>1.10 % Endlosschleife, nur mit <ESC>-Taste abzubrechen
>1.20 WHILE 1=1; DO 2; WAIT-T 60;
>1.30 END

>2.10 TYPE ! 'TK9QD11: '
>2.20 SET stat = TK9QD11(STATUS)
>2.30 TYPE ?stat
>2.40 RETURN
```

Das funktioniert folgendermaßen. Das Hauptprogramm besteht im Prinzip nur aus der Zeile `1.20`. Solange 1 = 1 ist (`WHILE 1=1`) – und das wird wohl noch einige Zeit so sein – werden die weiteren Befehle, die in der gleichen Zeile stehen, ausgeführt. Also, führe das Unterprogramm – die Gruppe 2 – aus (`DO 2`) und warte anschließend 60 Sekunden (`WAIT-T 60`) bis zur Ausführung des nächsten Schleifendurchlaufes.

Das geht solange gut, bis wir vom Gerät eine Fehlermeldung bekommen. Angenommen, es ist jemand über das Devicebus-Kabel gestolpert und hat es aus der Interfacekarte herausgerissen. Der Magnet ist somit nicht mehr erreichbar und es kommt zu einem Fehler. Dieser wird ausgegeben, NODAL bricht das Programm ab und geht zurück in den Direkt-Modus.

```
>RUN

TK9QD11: 11111111111101111110111111111110
TK9QD11: 11111111111101111110111111111110
TK9QD11: 11111111111101111110111111111110

***  NODAL ERROR  76   Error in USR response
%XSR-E-DEV_OFFLINE, addressed device is offline
>
```

[11] Wir möchten hier *ausdrücklich* darum bitten, keine sogenannten „Nudelprogramme“ zu schreiben. Nudelprogramme sind Programme, die – meist in einer Endlosschleife – ständig und ohne Pause etwas tun. Nudelprogramme nutzen die gesamte zur Verfügung stehende Rechenzeit eines Computers, so dass Programme, die eine niedrigere Priorität als NODAL haben (z.B. die „Batch Jobs“) überhaupt nicht mehr abgearbeitet werden können. Das macht die System Manager immer *etwas* böse! Um das zu vermeiden, benutzen wir in unserer Endlosschleife den Befehl „WAIT-T 60“. So macht NODAL nach jedem Durchlauf der Schleife eine Zwangspause von 60 Sekunden, in der die anderen Programme zu ihrem Recht kommen.

Wenn wir möchten, dass das Programm in diesem Fall nicht abbricht, können wir den Fehler im Programm selbst behandeln. Dazu schreiben wir unser Unterprogramm etwas um:

```
>2.10 TYPE ! 'TK9QD11: '
>2.20 SET stat = TK9QD11(STATUS, fehler)
>2.30 IF fehler = 0; TYPE ?stat; RETURN
>2.40 TYPE 'Fehler'
>2.50 RETURN
```

Das Wichtigste bei unserer Programmumstellung ist die Variable `fehler` in der Formulierung des Gerätezugriffs. Taucht hinter der Property, durch Komma getrennt, eine Variable auf, so unterbricht NODAL den Programmablauf *nicht*, wenn ein Fehler während des Gerätezugriffs auftritt, sondern weisst dieser Variablen die entsprechende NODAL-Fehlernummer zu. Eine Zuweisung erfolgt in jedem Fall. Trat kein Fehler auf, so enthält die Variable eine Null. In unserem Fehlerfall würde der Variablen `fehler` also die Nummer 76 (siehe oben) zugewiesen.

Unser Programm funktioniert nun folgendermaßen: Zunächst wird die Nomenklatur ausgegeben (Zeile `2.10`). Dann wird der Gerätezugriff mit Hilfe der Variablen `fehler` zur Aufnahme der Fehlernummer und der Variablen `stat` zur Aufnahme des Status gemacht (Zeile `2.20`). Trat kein Fehler auf (`fehler = 0`), so ist die Bedingung der `IF`-Anweisung wahr und die Befehle hinter der `IF`-Anweisung werden ausgeführt. Also wird der Status ausgegeben und anschließend das Unterprogramm verlassen (Zeile `2.30`). Trat ein Fehler auf (`fehler <> 0`), so werden die Befehle hinter der `IF`-Anweisung nicht durchlaufen, sondern sofort die nächste Zeile begonnen. Hier wird einfach nur „`Fehler`“ ausgegeben und das Unterprogramm anschließend ebenfalls verlassen. Die Ausgabe sieht dann folgendermaßen aus:

```
>RUN

TK9QD11: 11111111111101111110111111111110
TK9QD11: 11111111111101111110111111111110
TK9QD11: 11111111111101111110111111111110
TK9QD11: Fehler
TK9QD11: Fehler
TK9QD11: 11111111111101111110111111111110
```

Nun wird unser Programm beim Auftreten eines Fehlers nicht mehr abgebrochen, sondern einfach nur der Text „`Fehler`“ anstelle des Status ausgegeben. Hat der Gestolperte das Kabel wieder auf die Interfacekarte gesteckt, so kann es ohne Unterbrechung des Programms mit Statuslesen weitergehen.

Nun sagt natürlich „`Fehler`“ nicht sehr viel aus. Wir möchten gerne wissen, welcher Fehler aufgetreten ist. Wir könnten einfach die Fehlernummer mit ausgeben, aber die sagt auch nicht viel mehr aus, wenn man sie nicht zu interpretieren weiß.

Die Aufgabe der Interpretation kann man NODAL überlassen. Mit der Funktion `ERMES` wird der zur Fehlernummer dazugehörige Text ausgegeben. Unser erneut geändertes Unterprogramm sieht danach so aus:

```
>2.10 TYPE ! 'TK9QD11: '
>2.20 SET stat = TK9QD11(STATUS, fehler)
>2.30 IF fehler = 0; TYPE ?stat; RETURN
>2.40 TYPE ERMES(fehler)
>2.50 RETURN
```

Unser Unterprogramm hat sich nur in der Zeile `2.40` leicht geändert. Anstatt nur „`Fehler`“ auszugeben, übergeben wir der Funktion `ERMES` die Nummer des aufgetretenen Fehlers mit Hilfe unserer Variablen `fehler`. `ERMES` liefert den dazugehörigen Text, den wir sofort ausgeben.

Die entsprechende Ausgabe sieht dann folgendermaßen aus:

```
>RUN

TK9QD11: 11111111111101111110111111111110
TK9QD11: 11111111111101111110111111111110
TK9QD11: 11111111111101111110111111111110
TK9QD11: Error in USR response
TK9QD11: Error in USR response
TK9QD11: 11111111111101111110111111111110
```

Nun wissen wir also, dass die Gerätesoftware (die USR) mit einer Fehlermeldung geantwortet hat. Mit *welchem* Fehler sie geantwortet hat, lässt sich auch herausfinden. Bevor wir aber daran gehen, noch eine Vorüberlegung.

Die zwei für Gerätezugriffe wichtigsten NODAL-Fehlermeldungen sind die Meldungen mit den Nummern 75 und 76. Beide haben wir bereits kennengelernt.

```
***  NODAL ERROR  75   Error during USERFACE call
***  NODAL ERROR  76   Error in USR response
```

Immer dann, wenn wir eine Fehlermeldung vom Userface oder von der Gerätesoftware erhalten, wird von NODAL zusätzlich die entsprechende NODAL-Fehlermeldung mitgeliefert. Damit können wir nun einfach einen Gerätezugriff mit Fehlervariable formulieren. Den Wert der Variablen benutzen wir dazu, die weitere Fehlerverarbeitung zu bestimmen.

Unser Unterprogramm wollen wir zunächst einmal so erweitern, dass wir sehen, welcher Fehler von der Gerätesoftware gemeldet wurde. Das sieht dann so aus:

```
>2.10 TYPE ! 'TK9QD11: '
>2.20 SET stat = TK9QD11(STATUS, fehler)
>2.30 IF fehler = 0; TYPE ?stat; RETURN
>2.40 TYPE ERMES(fehler)
>2.50 IF fehler = 76; DO 2.95
>2.60 RETURN
>2.94 % Hier beginnen die Unterprogrammzeilen
>2.95 TYPE ! &9; TYPE GETMSG1(USRCFACI, USRCSTAT)
```

Neu in unserem Programm sind die Zeilen `2.50`, `2.94` und `2.95`. Zeile `2.94` ist eine reine Kommentarzeile. Zeile `2.50` ruft einfach die Zeile `2.95` auf, wenn ein Fehler in der Gerätesoftware aufgetreten ist (`fehler = 76`). War es ein anderer Fehler, ist das Unterprogramm zu Ende (`2.60 RETURN`).

Zeile `2.95`: Jede Gerätesoftware (jede USR) liefert bei jedem Zugriff einen „Completion Status“ und einen „Secondary Status“ zurück. Diese werden von NODAL intern in „Facility“ und den Status selbst aufgetrennt[12] und bis zum nächsten Gerätezugriff aufgehoben, wo sie von den neuen Status überschrieben werden. Den Completion Status, der der wichtigere der beiden ist, bekommt man über die Funktionen `USRCFACI` und `USRCSTAT`, und den Secondary Status, der nur selten versorgt wird, über die Funktionen `USRSFACI` und `USRSSTAT`. Die Funktion `GETMSG1` interpretiert diesen Wert (in unserem Fall nur den Completion Status) und liefert den entsprechenden Text dafür zurück, den wir ausgeben. Vorher haben wir noch einen Zeilenvorschub gemacht und eine Einrückung von 9 Leerzeichen (`&9`). Das Ergebnis sieht nun folgendermaßen aus:

[12] Die Auftrennung in Facility und Status ist historisch bedingt und existiert im Moment nur noch aus Gründen der Kompatibilität.

```
>RUN

TK9QD11: 11111111111101111110111111111110
TK9QD11: 11111111111101111110111111111110
TK9QD11: 11111111111101111110111111111110
TK9QD11: Error in USR response
         %XSR-E-DEV_OFFLINE, addressed device is offline
TK9QD11: Error in USR response
         %XSR-E-DEV_OFFLINE, addressed device is offline
TK9QD11: 11111111111101111110111111111110
```

Nun wissen wir also, dass der Fehler von einem nicht mehr erreichbaren (offline) Gerät herrührt. „Elektronisch“ gesprochen heißt das, dass die Interfacekarte am Gerät nicht mehr ansprechbar ist, was unter anderem auch an einem herausgerissenen Devicebus-Kabel liegen kann. Damit ist schon eine recht gute Eingrenzung des Fehlers möglich.

Auch Userface liefert bei jedem Zugriff einen Status zurück, der von NODAL intern bis zum Auftreten des nächsten Fehlers aufgehoben wird. Dieser Status lässt sich mit der Funktion `UFCLSTEX` direkt in Klartext ausgeben.

Zum letzten Mal wollen wir dazu unser Unterprogramm etwas erweitern.

```
>2.10 TYPE ! 'TK9QD11: '
>2.20 SET stat = TK9QD11(STATUS, fehler)
>2.30 IF fehler = 0; TYPE ?stat; RETURN
>2.40 TYPE ERMES(fehler)
>2.50 IF fehler = 76; DO 2.95
>2.60 IF fehler = 75; DO 2.96
>2.70 RETURN
>2.94 % Hier beginnen die Unterprogrammzeilen
>2.95 TYPE ! &9; TYPE GETMSG1(USRCFACI, USRCSTAT)
>2.96 TYPE ! &9; UFCLSTEX
```

In Zeile `2.60` wird getestet, ob es sich um einen Userface-Fehler handelt. Wenn ja, wird die Zeile `2.96` ausgeführt, die den Klartext des Userface-Fehlers ausgibt.

Angenommen, der ganze VME-Rahmen ist ausgefallen, so bekommen wir einen Fehler bereits vom Userface. Das sieht dann so aus:

```
>RUN

TK9QD11: 11111111111101111110111111111110
TK9QD11: 11111111111101111110111111111110
TK9QD11: 11111111111101111110111111111110
TK9QD11: Error during USERFACE call
         %UFC-E-DEVICE_OFFLINE, an addressed device is off-line
TK9QD11: Error during USERFACE call
         %UFC-E-DEVICE_OFFLINE, an addressed device is off-line
TK9QD11: 11111111111101111110111111111110
```

Kommt eine Meldung „`device offline`“ bereits vom Userface, kann man recht sicher auf einen Fehler im VME-Rahmen schließen. Letztenendes kann einfach die Power ausgeschaltet sein.

Bisher haben wir nur die Formulierung mit Einzelwert-Properties behandelt. Alle anderen Formulierungsarten funktionieren aber ganz analog: Die Fehler-Variable wird unmittelbar hinter der Property angegeben. Wir wollen die Formulierungen hier nur noch kurz aufzeigen.

Die Formulierung bei Mehrwert-Properties:

```
>1.10 CALL TK1MU1('R', devcon, CONSTANT, fehler)
```

Die Formulierung bei Properties ohne Datenwerten:

```
>1.10 CALL TK1MU1(RESET, fehler)
```

Die Formulierung bei Properties mit Parametern:

```
>1.10 SET XY_DEV(PROP(0), fehler)
>1.20 CALL XY_DEV(RESET(0), fehler)
>1.30 CALL S11MU1('R', i_ramp, CYCRMPSC(300, 399), fehler)
```

Man sollte jedoch tunlichst darauf achten, dass man die Fehlervariable nicht `ERROR` nennt! `ERROR` ist eine in NODAL vordefinierte Variable, deren Benutzung im Zusammenhang mit Gerätezugriffen zu sehr eigenartigen Fehlern führen kann.

12.2 Fehlerbehandlung mit dem Befehl DO

Eine weitere Behandlung von Fehlern ist in NODAL mit dem Befehl `DO` möglich. Diese Variante wird oft bei bewusst herbeigeführten Fehlern benutzt. Darauf gehen wir gleich in einem Beispiel ein. Zunächst die allgemeine Formulierung:

```
1.10 DO 10 !11 !12 !
1.20 ...
```

Zeile `1.10` zeigt die allgemeine Formulierung des Befehls `DO`, die wie folgt gelesen wird: „Führe die Programmgruppe `10` aus. Falls darin ein Fehler auftritt, verlasse die Gruppe und führe die Programmgruppe `11` aus. Falls darin ein Fehler auftritt, verlasse die Gruppe und führe die Programmgruppe `12` aus. Falls darin ein Fehler auftritt, verlasse die Gruppe und fahre mit der nächsten Zeile, also mit `1.20` fort.“ Natürlich sind auch kürzere Formulierungen möglich, wie zum Beispiel

```
1.10 DO 5 !
1.20 DO 10 !11
```

Man liest wie folgt. Zeile `1.10`: „Führe die Programmgruppe `5` aus. Falls darin ein Fehler auftritt, verlasse die Gruppe und fahre mit Zeile `1.20` fort.“ Zeile `1.20`: „Führe die Programmgruppe `10` aus. Falls darin ein Fehler auftritt, verlasse die Gruppe und führe die Programmgruppe `11` aus.“ Falls allerdings in der Programmgruppe `11` wiederum ein Fehler auftritt, wird das gesamte Programm unterbrochen und NODAL kehrt in den Direkt-Modus zurück.

Soweit zu den Möglichkeiten. Nun aber zu einem Beispiel, das schon in vielen NODAL-Programmen so oder ähnlich realisiert ist. Das Programm fragt den Benutzer nach einer Nomenklatur und gibt gleichzeitig die zuletzt gesetzte Nomenklatur als Default-Wert an. War noch keine Nomenklatur gesetzt, wird kein Default-Wert angegeben.

```
1.10 DO 19 !20
1.20 TYPE 'I =' %7.02 $nomen(CURRENTI) ' Ampere'
1.30 GOTO 1.10

19.05 TYPE ! 'Nomenklatur [' nomen ']'
19.10 $ASK newnom
19.15 $IF newnom <> ''; SET nomen = newnom
19.20 RETURN

20.10 TYPE ! 'Nomenklatur'
20.20 $ASK nomen
20.30 RETURN
```

Auch hier wollen wir nochmals auf die Formulierung `$nomen` im Gerätezugriff hinweisen, die an-

stelle der Nomenklatur eines Gerätes dort steht. Diese Art der Formulierung haben wir bereits kennengelernt, es ist eine „Indirection“.

Angenommen, wir haben gerade das Programm gestartet und kommen zum ersten Mal an der Zeile `1.10` vorbei. Dort wird zunächst die Gruppe `19` aufgerufen. In dieser Gruppe passiert schon in der ersten Zeile (`19.05`) ein Fehler, weil die Variable `nomen`, die wir ausgeben wollen, überhaupt noch nicht definiert ist. Also wird die Bearbeitung dieser Gruppe abgebrochen und die Gruppe `20` aufgerufen. Dort wird nach der Nomenklatur gefragt, die der Variablen `nomen` zugewiesen wird. Damit ist die Variable `nomen` definiert. Aus der Gruppe `20` kommen wir zurück in die Zeile `1.20`, wo der Iststrom des angegebenen Gerätes gelesen und ausgegeben wird.

Das würde am Bildschirm bis jetzt so aussehen:

```
>RUN

Nomenklatur: TK1MU1
I = 123.45 Ampere
```

Nach der Ausgabe geht's über Zeile `1.30` wieder zurück in die Zeile `1.10`. Dort wird zunächst wieder die Gruppe `19` aufgerufen. Nun ist die Variable `nomen` definiert und es passiert kein Fehler mehr. Wir werden nach einer neuen Nomenklatur gefragt (`$ASK newnom`) und bekommen als Vorschlag die alte Nomenklatur in eckigen Klammern, nämlich „`TK1MU1`“, angeboten. Antworten wir nur mit der Return-Taste (`newnom = ""`), so bleibt die alte Nomenklatur erhalten. Geben wir eine neue an (`newnom <> ""`), so wird `nomen` durch `newnom` überschrieben (`SET nomen = newnom`).

Insgesamt sieht das dann etwa so aus:

```
>RUN

Nomenklatur: TK1MU1
I = 123.45 Ampere

Nomenklatur [TK1MU1]: <RETURN>
I = 123.45 Ampere

Nomenklatur [TK1MU1]: S12MU3I
I = 4321.00 Ampere

Nomenklatur [S12MU3I]:
```

In diesem Zusammenhang ist auch die vordefinierte Variable `ERROR` von Interesse. `ERROR` enthält immer den zuletzt aufgetretenen NODAL-Fehler. In einer Gruppe, die aufgrund eines vorher aufgetretenen Fehlers aufgerufen wurde, kann man diese Variable testen, um den Fehler herauszubekommen und eine entsprechende Aktion einzuleiten. Ein kleines Beispiel hierzu soll dieses Kapitel beenden:

```
1.10 TYPE !!
1.20 ASK 'Zaehler' zaehler
1.30 ASK 'Nenner' nenner
1.40 DO 2 !
1.50 IF ERROR = 0; END
1.60 TYPE ! ERMES(ERROR)
1.70 IF ERROR = 6; TYPE ! 'Versuch durch Null zu teilen!'; GOTO 1.10
1.80 END

2.10 SET quotient = zaehler / nenner
2.20 TYPE quotient
2.30 RETURN
```

Die Interpretation überlassen wir jetzt dem mittlerweile geübten NODAListen.

13 Erweiterte Möglichkeiten mit Userface

Grundsätzlich gibt es in NODAL zwei Möglichkeiten des Zugriffs auf Geräte:

Datamodule Calls: Oder kurz DMS Calls. Diese wurden bereits ausführlich im Skript behandelt. Es sind die, von der Formulierung her, einfachsten Zugriffe auf Geräte. NODAL wandelt diese intern um, so dass auch Datamodule Calls letztenendes die Userface-Schnittstelle benutzen.

Userface-Schnittstelle: Userface ist eine eigenständige Schnittstelle zwischen Benutzerprogrammen auf der einen Seite und dem Kontrollsystem auf der anderen. Es stellt dem Benutzerprogramm Funktionen (Unterprogramme) zur Verfügung, die sehr flexible Möglichkeiten des Gerätezugriffes erlauben. Im Sinne des Userface ist NODAL selbst ein Benutzerprogramm wie jedes andere, das in FORTRAN, PASCAL oder anderen Sprachen geschrieben sein kann. NODAL wiederum stellt dem Benutzer Funktionen zur Verfügung, die es ihm erlauben, die Userface-Schnittstelle quasi direkt zu benutzen. Somit stehen auch dem NODAL-Benutzer sehr viele (nicht alle) Möglichkeiten dieser Schnittstelle zur Verfügung.

Um die Unterschiede zwischen einem Datamodule Call und der Benutzung einer Funktion des Userface kurz zu verdeutlichen, soll einmal ein Gerätezugriff via Datamodule Call und ein Zugriff via Userface, die exakt das gleiche tun, gegenübergestellt werden. Die Aufgabe sei, den Strom des Magneten `S12MU3I` auf 500 Ampere zu setzen.

Das geht via Datamodule Call ganz einfach so:

```
>SET S12MU3I(CURRENTS)=500
```

Via Userface wird's etwas komplizierter, weil hier viele Parameter versorgt werden müssen, bevor ein korrekter Gerätezugriff möglich wird:

```
>DIMENS-I info(5)
>DIMENS para(0)
>DIMENS-S func
>DIMENS data(1)
>DIMENS-I usrcst(4)
>$SET func(1) = 'S12MU3I'
>$SET func(2) = 'CURRENTS'
>$SET func(3) = 'W'
>SET data(1) = 500
>SET info(2) = 0
>SET info(4) = 1
>UFCEQACC(func, info, para, data, usrcst, 0, 0)
```

`UFCEQACC` – was soviel heißt wie „Userface equipment access“, also etwa „Userface Gerätezugriff“ – ist dabei der Aufruf einer Funktion der Userface-Schnittstelle. Alle Befehle zum Aufruf einer Funktion des Userface beginnen in NODAL mit `UFC`.

14 Funktionen

NODAL enthält über hundert Funktionen (Befehle), die in verschiedene Gruppen aufgeteilt sind. Wir wollen hier die Gruppen kurz vorstellen, aber nicht tiefer auf sie eingehen.

14.1 NODAL-Funktionen

Dahinter verstecken sich noch einmal verschiedene Untergruppen von Funktionen, die unter diesem Oberbegriff zusammengefasst wurden.

Mathematische Funktionen: Damit sind die üblichen Funktionen wie Sinus (`SIN`), Absolutwert (`ABS`) oder Quadratwurzel (`SQR`), aber auch boolsche Funktionen wie Schnittmenge (`AND`) oder Negation (`NEG`) gemeint.

Zeichenketten-Funktionen: Alle Funktionen, die mit Zeichenketten arbeiten, sind hierunter zusammengefasst. Es sind solche Funktionen wie die zur Bestimmung des ASCII-Wertes eines Zeichens (`ASCII`), zur Suche einer Zeichenkette in einer anderen (`FINDS`) und viele andere.

Pattern-Funktionen: Pattern sind eine Erweiterung der normalen Zeichenketten. Sie können sogenannte Wildcards enthalten, aus mehreren Zeichenketten gleichzeitig bestehen und einiges mehr. Die wichtigsten Befehle zur Konstruktion und zum Testen von Pattern sind `$PATTE` und `$MATCH`. NODAL ist sehr mächtig und vielseitig, was die Bearbeitung von Pattern betrifft, so dass wir hier nur auf die weiterführende Literatur hinweisen können ([4]).

Andere Funktionen: Hier findet man alle sonstwo nicht einzuordnenden Funktionen wie zum Beispiel die Funktion zur Ermittlung der Größe eines Feldes (`ARSIZE`), eine Funktion, die die Ziffern von 0 bis 9 zurückliefert (`NUM`) und unsere bereits bekannte Funktion `ERMES`.

14.2 Input/Output-Funktionen

NODAL bietet die Möglichkeit zu sogenanntem File-I/O. Das heißt, man kann eine Datei – ein File – anlegen und anschließend in diese Datei schreiben oder von dieser Datei lesen. Auch bereits bestehende Dateien können in dieser Art und Weise behandelt werden. Das ist sehr nützlich, wenn man zum Beispiel in einem Überwachungsprogramm für Geräte Protokoll führen möchte. Tritt ein Fehler auf, so kann man einfach die Fehlermeldung in eine Datei schreiben.

14.3 VMS-Funktionen

Das VAX-Betriebssystem VMS bietet einige Hilfen an, die auch dem Benutzer von NODAL von Nutzen sein können. Daher sind einige wenige VMS-Funktionen in NODAL implementiert. Eine haben wir bereits kennengelernt. Es ist die Funktion `GETMSG1`, die auf einen Fehlerkode den zugehörigen Klartext zurückliefert. Erwähnt sei noch die Funktion `TIME`, die die Anzahl der vergangenen Sekunden seit Mitternacht zurückliefert.[13]

[13] Eine schöne Übung in NODAL ist es, die Sekunden in die aktuelle Zeit in Stunden, Minuten und Sekunden umzurechnen und sie formatiert (etwa `HH:MM:SS`) auszugeben.

14.4 Datenbasis-Funktionen

So wie es für die Zugriffe auf Geräte die Schnittstelle des Userface gibt, so gibt es auch eine Schnittstelle, die Funktionen für Zugriffe auf die Datenbasis des Kontrollsystems, in der alle Geräte und viele weitere Geräte-Informationen enthalten sind, zur Verfügung stellen. Die Befehle zum Aufruf einer Funktion der Schnittstelle zur Datenbasis beginnen in NODAL mit `DBS`.

15 NODAL-eigene Werkzeuge

15.1 Der Editor in NODAL

Außer dem bereits erwähnten „Language Sensitive Editor“ (`LSEDIT`) auf der VAX besitzt NODAL einen eigenen „eingebauten“ Editor. Dieser bietet zwar bei weitem nicht die Möglichkeiten des LSE, ist aber auf der anderen Seite an die Erfordernisse, die beim Schreiben eines NODAL-Programmes auftreten, optimal angepasst. So erlaubt er zum Beispiel das automatische Erzeugen von Zeilennummern oder die Renummerierung von Programmblöcken. Dieser Editor wird mit dem Befehl `EDIT` aufgerufen. Nähere Informationen zu diesem Editor findet man in [2].

15.2 Der Debugger in NODAL

Programme sind oft sehr schwer von ihren letzten Fehlern zu befreien, besonders dann, wenn sie nicht mehr sehr übersichtlich sind, was gerade bei etwas größer werdenden NODAL-Programmen leicht geschehen kann. Ist ein Programm von allen Syntaxfehlern befreit, so können sich immer noch logische Fehler darin befinden. Das Programm stürzt dann zwar nicht mehr ab, es verhält sich aber nicht so, wie man eigentlich erwartet hätte. Hilfen zum Herausfinden von Fehlern, insbesondere von logischen, bietet der in NODAL integrierte Debugger (zu deutsch etwa „Entwanzer“). Er erlaubt unter anderem, in Programme automatische Unterbrechungen (sogenannte „Breakpoints“) einzubauen, die das ablaufende Programm an bestimmten benutzerdefinierten Stellen anhalten. Dies ermöglicht dem Programmentwickler ausführliche Tests seiner Software.

Eine ausführliche Beschreibung des Debuggers und seiner Eigenschaften findet man in [6].

A Liste der NODAL-Fehler

Dies ist die Liste aller möglichen Fehlermeldungen von NODAL. Angegeben sind die Fehlernummern und die zugehörigen Fehlertexte.

Nr.	Text	Nr.	Text
0	No error	1	Illegal line number
2	Illegal format specifier	3	Illegal arithmetic expression
4	Ambigious command	5	Illegal delimiter
6	Attempt to divide by zero	7	Working area full
8	Nonexistent name	9	Wrong variable type
10	Link resources exhausted	11	Command not properly terminated
12	Unallocated error	13	Nonexistent line addressed
14	Illegal shuffle attempted	15	Error in IF command
16	Escape typed	17	Illegal edit command
18	Illegal ASK command	19	Erase error
20	Argument list error	21	File error
22	Error in SAVE command	23	Array dimension error
24	Square root of negative number	25	Illegal arctangent argument
26	Sine argument too big	27	Cosine argument too big
28	Power error [negative argument?]	29	Power underflow
30	Exponential argument too big	31	Logarithm argument <= 0
32	Device not connected	33	Unauthorised action
34	Hardware error	35	Illegal equipment number
36	Illegal property	37	Value out of range
38	Not implemented	39	No such computer
40	Result string filled	41	Syntax error
42	No such file	43	File already exists
44	No file space	45	Link not open
46	Remitted data lost	47	End of file
48	Equipment error	49	SIOM error
50	Illegal error number	51	Checksum error
52	Defined function area full	53	Syntax error in DEFINE command
54	Illegal string in SET command	55	String function failure
56	Illegal concatenation	57	Error in $IF command
58	Error in $ASK command	59	String expected

... Fortsetzung auf der nächsten Seite

Nr.	Text	Nr.	Text
60	Pattern too big	61	Bad pattern match
62	Bad pattern	63	Bad pattern assignment
64	Indirection signal	65	not used
66	not used	67	not used
68	To many nested DO	69	not used
70	not used	71	Unknown terminal
72	Channel transfer error	73	Breakpoint found
74	Error during DMS call	75	Error during USERFACE call
76	Error in USR response	77	Error in DBS functions

Literatur

[1] H. Hübner. *Gerätezugriffe mit NODAL*. SIS/ESR Controls Documentation U-NOD-01.

[2] L. Hechler, H. Hübner, U. Krause. *The NODAL Interpreter at GSI*. SIS/ESR Controls Documentation U-NOD-02.

[3] K. H. Lundberg. *NODAL Primer*. SPS/ACC/Note/85-27.

[4] P. D. V. van der Stok. *NODAL Reference Manual*. SPS/ACC/Note/85-33.

[5] M. C. Crowley-Milling and G. C. Shering. *The NODAL System for the SPS*. CERN 78-07.

[6] A. Lacroix. *NODAL Debugger User's Guide*. LEP Controls Note 78. SPS/ACC/Note/86-23.

Index

— **T** —

— **U** —

— **V** —

— **W** —

— **Z** —