

STACK

```
STACK [TOP] { COMMAND operand1 [operand2 [(parameter)]] ... }
              { [DATA] [FORMATTED] { operand2 [(parameter)]] ... }
```

Dieses Kapitel behandelt folgende Themen:

- Funktion
- Syntax-Beschreibung
- Beispiel

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

Verwandte Statements: INPUT | RELEASE

Funktion

Das Statement STACK dient dazu, Daten im Natural-Stack abzulegen. Hierbei kann es sich um folgende Daten handeln:

- den Namen eines Natural-Programms oder -Systemkommandos, das ausgeführt werden soll;
- Daten, die bei der Ausführung eines INPUT-Statements als Eingabedaten verwendet werden sollen.

Weitere Informationen zum Stack finden Sie im Kapitel *Weitere Programmieraspekte, Stack im Leitfaden zur Programmierung*.

Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur					Mögliche Formate										Referenzierung erlaubt	Dynam. Definition							
<i>operand1</i>	C	S	A	G	N	A																	ja	ja
<i>operand2</i>	C	S	A	G	N	A	U	N	P	I	F	B	D	T	L	G							ja	ja

Syntax-Element-Beschreibung:

TOP	<p>Normalerweise werden die Daten unten im Stack abgelegt. Das Schlüsselwort TOP bewirkt, dass die Daten oben auf dem Natural Stack abgelegt werden.</p> <p>Beispiel: Mit diesem Statement wird der Inhalt der Variablen #FIELDA oben auf dem Stack abgelegt:</p> <pre>STACK TOP #FIELDA</pre>
DATA	<p>Mit DATA (Standardeinstellung) legen Sie Daten im Stack ab, die von einem INPUT-Statement als Eingabedaten verwendet werden.</p> <p>Begrenzungszeichen oder <i>Input Assign</i>-Zeichen innerhalb der übergebenen Datenwerte werden als Begrenzung interpretiert und entsprechend verarbeitet. Einzelheiten darüber, wie die Daten von einem INPUT-Statement verarbeitet werden, können Sie der Beschreibung des INPUT-Statements <i>Eingabedaten aus dem Natural-Stack</i> entnehmen.</p> <p>Beispiel: Mit den folgenden Statements wird der Inhalt der Variablen #FIELD1 und #FIELD2 im Stack abgelegt:</p> <pre>MOVE 'ABC' TO #FIELD1 MOVE 'XYZ' TO #FIELD2 STACK #FIELD1 #FIELD2</pre> <p>Diese Variablen werden als Eingabedaten an das nächste INPUT-Statement im Natural-Programm übergeben, und zwar im Begrenzungs-Modus:</p> <pre>INPUT #FIELD1 #FIELD2</pre> <p>Anmerkung: Wenn <i>operand2</i> eine Zeitvariable (Format T) ist, wird nur die Zeitkomponente des Variableninhalts im Stack abgelegt, aber nicht die Datumskomponente.</p>

<p>FORMATTED</p>	<p>Das Schlüsselwort FORMATTED bewirkt, dass alle Daten Feld für Feld an das nächste INPUT-Statement übergeben werden. Schlüsselzuordnungen oder Begrenzungszeichen werden nicht als solche interpretiert.</p> <p>Beispiele:</p> <p>Mit den folgenden Statements wird ABC , DEF in #FIELD1 übertragen und XYZ in #FIELD2:</p> <pre>MOVE 'ABC,DEF' TO #FIELD1 MOVE 'XYZ' TO #FIELD2 STACK TOP DATA FORMATTED #FIELD1 #FIELD2 ... INPUT #FIELD1 #FIELD2</pre> <p>Angenommen, das Input-Begrenzungszeichen ist das Komma (Profil-/Session-Parameter ID= ,), dann wird mit folgenden Statements — ohne das Schlüsselwort FORMATTED — ABC in #FIELD1 übertragen und DEF in #FIELD2:</p> <pre>MOVE 'ABC,DEF' TO #FIELD1 STACK TOP DATA #FIELD1 ... INPUT #FIELD1 #FIELD2</pre> <p>Anmerkung: Die FORMATTED-Option sollte verwendet werden, wenn die zu übergebenden Daten Begrenzungs-, Steuer- oder DBCS-Zeichen enthalten, um eine unbeabsichtigte Interpretation dieser Zeichen zu vermeiden.</p>
<p>COMMAND <i>operand1</i></p>	<p>Um ein Kommando (bzw. einen Programmnamen) im Stack abzulegen, geben Sie das Schlüsselwort COMMAND gefolgt von dem betreffenden Kommando (<i>operand1</i>) an. Würde ein Programm normalerweise den Benutzer mit einer Eingabeaufforderung in Form einer NEXT-Zeile konfrontieren, so unterdrückt nun Natural die Anzeige der NEXT-Zeile und führt stattdessen das im Stack abgelegte Kommando aus.</p> <p>Beispiel: Mit dem folgenden Statement wird das Kommando RUN oben auf dem Stack abgelegt. Natural führt dieses Kommando aus, wenn normalerweise das nächstmal die NEXT-Zeile ausgegeben würde:</p> <pre>STACK TOP COMMAND 'RUN'</pre>

<p>COMMAND <i>operand1 operand2</i> ...</p>	<p>Zusammen mit einem Kommando (<i>operand1</i>) können Sie auch Daten (<i>operand2</i>) im Stack ablegen. Diese Daten werden dann vom nächsten INPUT-Statement nach der Ausführung des Kommandos als Eingabedaten verarbeitet.</p> <p>Zusammen mit einem Kommando abgelegte Daten werden immer unformatiert abgelegt.</p> <p>Anmerkung: Wenn in den abzulegenden Daten leere alphanumerische Felder (d.h. Leerzeichen) enthalten sind, werden diese Leerzeichen als Delimiter zwischen Werten interpretiert und folglich von dem betreffenden INPUT-Statement falsch verarbeitet. Wenn Sie daher leere alphanumerische Felder als Daten zusammen mit einem Kommando im Stack ablegen möchten, müssen Sie hierzu zwei STACK-Statements verwenden: Ein STACK DATA <i>operand2 . . .</i>, um die Daten abzulegen, und ein STACK COMMAND <i>operand1</i>, um das Kommando abzulegen.</p>
<p><i>parameter</i></p>	<p>Wenn <i>operand2</i> eine Datumsvariable ist, können Sie den Session-Parameter DF (Datumsformat) als <i>parameter</i> für diese Variable angeben.</p>

Beispiel

```

** Example 'STKEX1': STACK
*****
DEFINE DATA LOCAL
1 #CODE (A1)
END-DEFINE
*
INPUT //
  10X 'PLEASE SELECT COMMAND' //
  10X 'LIST VIEW      (V)' /
  10X 'LIST PROGRAM * (P)' /
  10X 'TECH INFO     (T)' /
  10X 'STOP          (.)' //
  20X 'CODE:' #CODE
*
*
DECIDE ON FIRST #CODE
  VALUE 'V'
    STACK TOP DATA 'VIEW'
    STACK TOP COMMAND 'LIST'
  VALUE 'P'
    STACK TOP COMMAND 'LIST PROGRAM *'
  VALUE 'T'
    STACK TOP COMMAND 'LAST *'
    STACK TOP COMMAND 'TECH'
    STACK TOP COMMAND 'SYSPROD'
  VALUE '.'
    STOP
  NONE
  REINPUT 'PLEASE ENTER VALID CODE'

```

```
END-DECIDE
*
*
END
```

Ausgabe des Programms STKEX1:

PLEASE SELECT COMMAND

```
LIST VIEW      (V)
LIST PROGRAM * (P)
TECH INFO      (T)
STOP           (.)
```

CODE:P

Nach Eingabe und Bestätigung des Codes:

```
16:46:28          ***** NATURAL LIST COMMAND *****          2005-01-19
User HTR          - LIST Objects in a Library -                  Library SYSEXSYN
```

Cmd	Name	Type	S/C	SM	Version	User ID	Date	Time
---	*_____	P_____	*__	* *	_____	*_____	_____	*_____
___	ACREX1	Program	S/C	S	4.1.03	RKE	2004-11-11	16:32:37
___	ACREX2	Program	S/C	S	4.1.03	RKE	2005-01-05	10:29:51
___	ADDEX1	Program	S/C	S	4.1.03	RKE	2004-11-11	16:36:49
___	AEDEX1R	Program	S/C	R	4.1.03	RKE	2004-11-11	16:40:34
___	AEDEX1S	Program	S/C	S	4.1.03	RKE	2004-11-11	16:39:57
___	AEPEX1R	Program	S/C	R	4.1.03	RKE	2004-11-11	16:41:57
___	AEPEX1S	Program	S/C	S	4.1.03	RKE	2004-11-11	16:42:31
___	AEPEX2	Program	S/C	S	4.1.03	RKE	2004-11-11	16:43:37
___	ASDEX1R	Program	S/C	R	4.1.03	RKE	2004-11-11	17:00:21
___	ASDEX1S	Program	S/C	S	4.1.03	RKE	2004-11-11	17:00:50
___	ASGEX1R	Program	S/C	R	4.1.03	RKE	2004-11-11	17:02:01
___	ASGEX1S	Program	S/C	S	4.1.03	RKE	2004-11-11	17:02:08
___	ATBEX1R	Program	S/C	R	4.1.03	RKE	2004-11-11	17:03:18
___	ATBEX1S	Program	S/C	S	4.1.03	RKE	2004-11-11	17:03:05

14 Objects found

Top of List.

Command ==>

```
Enter-PF1---PF2---PF3---PF4---PF5---PF6---PF7---PF8---PF9---PF10--PF11--PF12---
      Help  Print Exit  Sort      --  -  +  ++      >  Canc
```