Benutzervariablen

Von Ihnen selbst in einem Programm definierte Variablen werden als Benutzervariablen bezeichnet. Sie können Sie dazu verwenden, in einem Programm Werte oder Zwischenergebnisse zu speichern, die Sie später weiterverarbeiten oder ausgeben möchten.

Dieses Kapitel behandelt folgende Themen:

- Definition von Benutzervariablen
- Datenbankfelder mit der (r)-Notation referenzieren
- Sourcecode-Zeilennummern umnummerieren
- Format und Länge von Benutzervariablen
- Spezielle Formate
- Index-Notation
- Datenbank-Array referenzieren
- Internen Zähler für ein Datenbank-Array referenzieren — C*-Notation
- Datenstrukturen qualifizieren
- Beispiele für Benutzervariablen

Siehe auch *Namenskonventionen für Benutzervariablen* in der Dokumentation *Natural benutzen*.

Definition von Benutzervariablen

Sie definieren eine Benutzervariable, indem Sie den Namen sowie das Format und die Länge der Variablen im `DEFINE DATA`-Statement angeben.

Sie definieren die Eigenschaften einer Variable mit der folgenden Notation:

```
(r,format-length/index)
```

Diese Notation folgt auf den Variablennamen, getrennt durch eine oder mehrere Leerstellen, als Option.

Zwischen den einzelnen Elementen der Notation sind keine Leerstellen zulässig.

Die einzelnen Elemente können erforderlichenfalls selektiv angegeben werden. Wenn sie aber zusammen benutzt werden, müssen Sie durch die oben angegebenen Zeichen voneinander getrennt werden.

Beispiel:

In diesem Beispiel ist eine Benutzervariable mit alphanumerischem Format und einer Länge von 10 Stellen unter dem Namen #FIELD1 definiert.

```
DEFINE DATA LOCAL
1 #FIELD1 (A10)
...
END-DEFINE
```

Anmerkungen:

1. Im Strucutured Mode oder wenn ein Programm eine DEFINE DATA LOCAL-Klausel enthält, können Variablen nicht dynamisch in einem Statement definiert werden.
2. Dies gilt nicht für anwendungsunabhängige Variablen (AIVs). Siehe auch *Anwendungsunabhängige Variablen definieren*.

Datenbankfelder mit der (*r*)-Notation referenzieren

Ein Statement-Label oder die Sourcecode-Zeilennummer kann zum Referenzieren eines vorher eingesetzten Natural-Statements benutzt werden. Damit kann die Standard-Referenzierung von Natural überschrieben werden (wie für jedes einzelne Statement beschrieben, wo zutreffend), oder es wird zu Dokumentationszwecken verwendet. Siehe auch *Schleifenverarbeitung*, Unterabschnitt *Statements innerhalb eines Programms referenzieren*.

Folgende Themen werden behandelt:

- Standard-Referenzierung von Datenbankfeldern
- Referenzieren mit Statement-Labels
- Referenzieren mit Sourcecode-Zeilennummern

Standard-Referenzierung von Datenbankfeldern

Im Allgemeinen gilt Folgendes, wenn Sie keine Statement-Referenzierung spezifizieren:

- Standardmäßig wird die innerste aktive Datenbank-Schleife (FIND, READ oder HISTOGRAM) referenziert, in der das betreffende Datenbankfeld eingelesen wurde.
- Wenn das Feld in keiner aktiven Datenbank-Schleife eingelesen wurde, wird das letzte vorher verwendete GET-Statement (im Reporting Mode auch FIND FIRST oder FIND UNIQUE-Statement) referenziert, welches das Feld eingelesen hat.

Referenzieren mit Statement-Labels

Ein Natural-Statement, das bewirkt, dass eine Verarbeitungsschleife initiiert wird und/oder dass Datenelemente in der Datenbank aufgerufen werden, kann mit einem symbolischen Label zur nachfolgenden Referenzierung versehen werden.

Ein Label kann entweder in der Form "*label.*" vor dem referenzierenden Objekt oder in Klammern "(*label.*)" nach dem referenzierenden Objekt (aber nicht beide gleichzeitig) angegeben werden.

Die Namenskonventionen für Labels sind identisch mit denen für Variablen. Der Punkt nach dem Label-Namen dient zur Identifizierung des Eintrags als ein Label.

Beispiel:

```
...
 RD. READ PERSON-VIEW BY NAME STARTING FROM 'JONES'
   FD. FIND AUTO-VIEW WITH PERSONNEL-ID = PERSONNEL-ID (FD.)
         DISPLAY NAME (RD.) FIRST-NAME (RD.) MAKE (FD.)
    END-FIND
 END-READ
 ...
```

Referenzieren mit Sourcecode-Zeilennummern

Ein Statement kann auch referenziert werden, wenn dafür die Nummer der Sourcecode-Zeile, in der das Statement sich befindet, benutzt wird.

Alle vier Ziffern der Zeilennummer müssen angegeben werden (führende Nullen dürfen nicht weggelassen werden).

Beispiel:

```
...
0110 FIND EMPLOYEES-VIEW WITH NAME = 'SMITH'
0120   FIND VEHICLES-VIEW WITH MODEL = 'FORD'
0130     DISPLAY NAME (0110) MODEL (0120)
0140   END-FIND
0150 END-FIND
...
```

Sourcecode-Zeilennummern umnummerieren

Vierstellige, numerische Sourcecode-Zeilennummern, die ein Statement referenzieren (siehe *Notation zur Statement-Referenzierung — (r)*), werden auch umnummeriert, wenn das Natural-Sourceprogramm umnummeriert wird. Zur Unterstützung des Benutzers und zur Verbesserung der Programm-Lesbarkeit und -Fehlerbeseitigung werden alle in einem Statement, einer alphanumerischen Konstante oder in einem Kommentar auftretenden Sourcecode-Zeilennummern umnummeriert. Die Position der Sourcecode-Zeilennummer im Statement oder der alphanumerischen Konstante (Anfang, Mitte, Ende) spielt dabei keine Rolle.

Die folgende Zeichenfolge wird als eine gültige Referenz auf eine Sourcecode-Zeilennummer erkannt und umnummeriert (*nnnn* ist eine vierstellige Ziffer):

Zeichenfolge	Beispiel-Statement
(*nnnn*)	`ESCAPE BOTTOM (0150)`
(*nnnn /*	`DISPLAY ADDRESS-LINE(0010/1:5)`
(*nnnn,*	`DISPLAY NAME(0010,A10/1:5)`

Wenn der linken Klammer oder der vierstelligen Ziffer *nnnn* eine Leerstelle folgt, oder wenn der vierstelligen Ziffer *nnnn* ein Punkt folgt, gilt die Zeichenfolge nicht als eine gültige Referenz auf eine Sourcecode-Zeilennummer.

Um zu vermeiden, dass eine vierstellige, in einer alphanumerischen Konstante enthaltene Ziffer aus Zufall umnummeriert wird, sollte die Konstante aufgeteilt werden, und die unterschiedlichen Bestandteile sollten mittels eines Bindestriches miteinander zu einem Wert verkettet werden.

Beispiel:

```
Z := 'XXXX (1234,00) YYYY'
```

sollte ersetzt werden durch

```
Z := 'XXXX (1234' - ',00) YYYY'
```

Format und Länge von Benutzervariablen

Das Format und die Länge einer Benutzervariablen werden in Klammern hinter dem Namen der Variablen angegeben.

Variablen fester Länge können eine/s der folgenden Formate und Längen haben.

Zur Definition von Format und Länge bei dynamischen Variablen siehe *Definition dynamischer Variablen*.

Format		Definierbare Länge	Interne Länge (in Bytes)
A	Alphanumerisch	1 - 1073741824 (1 GB)	1 - 1073741824
B	Binär	1 - 1073741824 (1 GB)	1 - 1073741824
C	Attribut-Zuweisung	-	2
D	Datum	-	4
F	Gleitkomma	4 oder 8	4 oder 8
I	Integer (Ganzzahl)	1 , 2 oder 4	1, 2 oder 4
L	Logisch	-	1
N	Numerisch (ungepackt)	1 - 29	1 - 29
P	Numerisch (gepackt)	1 - 29	1 - 15
T	Zeit	-	7
U	Unicode (UTF-16)	1 - 536870912 (0,5 GB)	2 - 1073741824

Die Länge kann nur angegeben werden, wenn das Format angegeben ist. Bei einigen Formaten braucht die Länge nicht explizit angegeben zu werden (wie in der Tabelle oben gezeigt).

Für mit Format N oder P definierte Felder können Sie die Dezimalstellen-Notation in der Form *nn.m* benutzen. *nn* stellt die Anzahl der Stellen vor dem Dezimalkomma dar, und *m* stellt die Anzahl der Stellen nach dem Dezimalkomma dar. Die Summe der Werte von *nn* und *m* darf 29 nicht überschreiten, und der Wert von of *m*darf 7 nicht überschreiten.

Die maximal definierbare Länge (1 GB für alphanumerische, binäre und Unicode-Felder) stellt die vom Natural-Compiler auferlegte Grenze dar. In Wirklichkeit ist der Speicherplatz, der als Datenspeicher bereitstehen kann, allerdings sehr viel kleiner. Insbesondere in einer "Natural Thread"-basierten

Umgebung ist die Größe der sessionabhängigen Benutzerbereiche, also der Umfang der Benutzerfelder in der Data Area auf den mit dem Schlüsselwort-Parameter MAXSIZE im Makro NTSWPRM definierten Wert beschränkt.

Anmerkungen:

1. Wenn eine Benutzervariable vom Format P mit einem DISPLAY-, WRITE- oder INPUT-Statement ausgegeben wird, wandelt Natural intern das Format P in das Format N für die Ausgabe um.
2. Wenn im Reporting Mode Format und Länge nicht für eine Benutzervariable angegeben werden, wird das/die Standard-Format/Länge N7 benutzt, es sei denn, diese Standard-Zuweisung wurde durch den Profil/Session-Parameter FS ausgeschaltet.

Für ein Datenbankfeld gilt das/die Format/Länge, wie für das Feld in der DDM definiert. (Im Reporting Mode ist es auch möglich, in einem Programm ein/e andere/s Format/Länge für ein Datenbankfeld zu definieren.)

Im Structured Mode kann Format und Länge nur in einer Data Area-Definition oder mit einem DEFINE DATA-Statement angegeben werden.

Beispiel für Format/Längen-Definition — Structured Mode:

```
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
1 #NEW-SALARY (N6.2)
END-DEFINE
...
FIND EMPLOY-VIEW ...
...
COMPUTE #NEW-SALARY = ...
...
```

Im Reporting Mode kann Format/Länge im Hauptteil des Programms definiert werden, wenn kein DEFINE DATA-Statement benutzt wird.

Beispiel für eine Format/Längen-Definition — Reporting Mode:

```
...
...
 FIND EMPLOYEES
... ... COMPUTE #NEW-SALARY(N6.2) = ...
...
```

Spezielle Formate

Zusätzlich zu den standardmäßigen Formaten alphanumerisch (A bzw. U) und numerisch (B, F, I, N, P) unterstützt Natural die folgenden speziellen Formate:

- Format C — Attribut-Kontrolle
- Formate D — Datum und T - Zeit

- Format L — Logisch
- Format Handle of Object

Format C — Attribut-Kontrolle

Eine mit dem Format C definierte Variable kann benutzt werden, um Attribute dynamisch einem in einem DISPLAY, INPUT-, PRINT-, PROCESS PAGE- oder WRITE-Statement benutzten Feld zuzuweisen.

Für eine Variable des Formats C kann keine Länge angegeben werden. Der Variable wird stets eine Länge von 2 Bytes von Natural zugewiesen.

Beispiel:

```
DEFINE DATA LOCAL
1 #ATTR (C)
1 #A (N5)
END-DEFINE
...
MOVE (AD=I CD=RE) TO #ATTR
INPUT #A (CV=#ATTR)
...
```

Weitere Informationen siehe Session-Parameter CV.

Formate D — Datum und T - Zeit

Mit den Formaten D und T definierte Variablen können für die Datums- und Zeit-Arithmetik und -Anzeige benutzt werden. Das Format D kann nur Datums-Informationen enthalten. Das Format T kan sowohl Datums- als auch Zeit-Informationen enthalten; mit anderen Worten sind Datums-Informationen eine Untermenge der Zeit-Informationen. Die Zeit wird in Zehntelsekunden gemessen.

Für Variablen der Formate D und T kann keine Länge angegeben werden. Einer Variable mit Format D wird stets eine Länge von 4 Bytes (P6) und einer Variable mit Format T wird stets eine Länge von 7 Bytes (P12) von Natural zugewiesen. Wenn der Profilparameter MAXYEAR auf 9999 gesetzt ist, weist Natural einer Variablen des Formats D immer eine Länge von 4 Bytes (P7) zu, und einer Variablen des Formats T eine Länge von 7 Bytes (P13).

Beispiel:

```
DEFINE DATA LOCAL
1 #DAT1 (D)
END-DEFINE
*
MOVE *DATX TO #DAT1
ADD 7 TO #DAT1
WRITE '=' #DAT1
END
```

Weitere Informationen siehe Session-Parameter EM und Systemvariable *DATX und *TIMX.

Der Wert in einem Datumsfeld muss im Bereich vom 1. Januar 1582 bis 31. Dezember 2699 liegen.

Format L — Logisch

Eine mit Format L definierte Variable kann als ein logisches Bedingungskriterium benutzt werden. Sie kann den Wert TRUE (wahr) oder FALSE (falsch) annehmen.

Für eine Variable des Formats L kann keine Länge angegeben werden. Einer Variable des Formats L wird stets eine Länge von 1 Byte von Natural zugewiesen.

Beispiel:

```
DEFINE DATA LOCAL
1 #SWITCH(L)
END-DEFINE
MOVE TRUE TO #SWITCH
...
IF #SWITCH
   ...
   MOVE FALSE TO #SWITCH
ELSE
   ...
   MOVE TRUE TO #SWITCH
END-IF
```

Weitere Informationen zur logischen Wertedarstellung siehe Session-Parameter EM.

Format Handle of Object

Eine als HANDLE OF OBJECT definierte Variable kann als Objekt-Handle benutzt werden.

Weitere Informationen siehe *NaturalX*.

Index-Notation

Eine Index-Notation wird für Felder benutzt, die ein Array darstellen.

Eine numerische Ganzzahl-Konstante oder Benutzervariable kann bei Index-Notationen eingesetzt werden. Eine Benutzervariable kann unter Benutzung von einem der folgenden Formate angegeben werden: N (numerisch), P (gepackt), I (Ganzzahl) oder B (binär), wobei das Format B nur mit einer Länge von kleiner oder gleich 4 benutzt werden kann.

Eine Systemvariable, Systemfunktion oder qualifizierte Variable kann nicht für Index-Notationen benutzt werden.

Array-Definition – Beispiele:

1. `#ARRAY (3)`

   Definiert ein eindimensionales Array mit drei Ausprägungen.

2. `FIELD (`*`label`*`.,A20/5)` oder *`label`*`.FIELD(A20/5)`

   Definiert ein Array von einem Datenbankfeld, das das durch *`label`* markierte Statement mit dem Format alphanumerisch, der Länge 20 und 5 Ausprägungen referenziert.

3. `#ARRAY (N7.2/1:5,10:12,1:4)`

   Definiert ein Array mit Format/Länge N7.2 und drei Array-Dimensionen mit 5 Ausprägungen (1 bis 5) in der ersten, 3 Ausprägungen (10 bis 12) in der zweiten und 4 Ausprägungen (1 bis 4) in der dritten Dimension.

4. `FIELD (`*`label`*`./i:i + 5)` oder *`label`*`.FIELD(i:i + 5)`

   Definiert ein Array von einem Datenbankfeld, das das durch *`label`*`.` markierte Statement referenziert.

   `FIELD` stellt ein multiples Feld oder ein Feld aus einer Periodengruppe dar, wobei *`i`* den Index für die relative Position innerhalb der Datenbank-Ausprägung angibt. Die Größe des Arrays innerhalb des Programms ist definiert als 6 Ausprägungen (i:i + 5). Der Datenbank-Index für die relative Position ist angegeben als eine Variable, um eine Positionierung des Programm-Arrays innerhalb der Ausprägungen des multiplen Feldes oder der Periodengruppe zu ermöglichen. Für eine erneute Positionierung von *`i`* muss über ein `GET`- oder `GET SAME`-Statement ein neuer Aufruf der Datenbank erfolgen.

Natural ermöglicht die Definition von Arrays, bei denen der Index nicht mit `1` anfangen muss. Zur Laufzeit überprüft Natural, ob in der Referenz angegebene Indexwerte nicht die maximale Größe der in der Definition spezifizierten Dimensionen überschreiten.

Anmerkungen:

1. Aus Kompatiblitätsgründen zu früheren Versionen von Natural kann ein Array-Bereich mittels eines Bindestrichs (-) anstatt eines Doppelpunkts (:) angegeben werden.
2. Eine Mischung aus beiden Notationen ist allerdings *nicht* zulässig.
3. Die Bindestrich-Notation ist nur zulässig im Reporting Mode (aber *nicht* in einem `DEFINE DATA`-Statement).

Der maximale Indexwert ist 1073741824. Die maximale Größe einer Data Area pro Programmierobjekt ist 1073741824 Bytes (1 GB).

Warnung:
Zur Kompatibilität mit der Natural Version 3.1 auf Großrechnern: Benutzen Sie die `V41COMP`-Kompilierungsoption des Profilparameters `CMPO` oder des Makros `NTCMPO`, um diese Beschränkungen aus Kompatibilitätsgründen auf die für die Natural Version 4.1 auf Großrechnern gültigen Grenzen zu reduzieren.

Einfache arithmetische Ausdrücke können mittels der Operatoren "+" und "-" in Index-Referenzen verwendet werden. Wenn arithmetische Ausdrücke als Indizes benutzt werden, muss den Operatoren "+" oder "-" ein Leerzeichen vorausgehen und ihnen folgen.

Arrays in Gruppen-Strukturen werden von Natural Feld für Feld aufgelöst, und nicht Gruppen-Ausprägung für Gruppen-Ausprägung.

Beispiel für die Auflösung von Gruppen-Arrays:

```
DEFINE DATA LOCAL
 1 #GROUP (1:2)
   2 #FIELDA (A5/1:2)
   2 #FIELDB (A5)
 END-DEFINE
 ...
```

Wenn die oben definierte Gruppe in einem `WRITE`-Statement ausgegeben würde:

```
WRITE #GROUP (*)
```

würden die Ausprägungen in der folgenden Reihenfolge ausgegeben:

```
#FIELDA(1,1) #FIELDA(1,2) #FIELDA(2,1) #FIELDA(2,2) #FIELDB(1) #FIELDB(2)
```

und *nicht*:

```
#FIELDA(1,1) #FIELDA(1,2) #FIELDB(1) #FIELDA(2,1) #FIELDA(2,2) #FIELDB(2)
```

Referenzieren von Arrays — Beispiele:

1. `#ARRAY (1)`

   Referenziert die erste Ausprägung eines eindimensionalen Arrays.

2. `#ARRAY (7:12)`

   Referenziert die siebente bis zwölfte Ausprägung eines eindimensionalen Arrays.

3. `#ARRAY (i + 5)`

   Referenziert die i+fünfte Ausprägung eines eindimensionalen Arrays.

4. `#ARRAY (5,3:7,1:4)`

   Es erfolgt eine Referenz innerhalb eines dreidimensionalen Arrays auf die Ausprägung 5 in der ersten Dimension, die Ausprägungen 3 bis 7 (5 Ausprägungen) in der zweiten Dimension und 1 bis 4 (4 Ausprägungen) in der dritten Dimension.

5. Stern-Notation (*) kann benutzt werden, um alle Ausprägungen innerhalb einer Dimension zu referenzieren:

   ```
   DEFINE DATA LOCAL
   1 #ARRAY1 (N5/1:4,1:4)
   1 #ARRAY2 (N5/1:4,1:4)
   END-DEFINE
   ...
   ADD #ARRAY1 (2,*) TO #ARRAY2 (4,*)
   ...
   ```

Benutzung eines Schrägstrichs vor einer Array-Ausprägung

Wenn auf einen Variablennamen eine vierstellige Zahl in Klammern folgt, interpretiert Natural diese Zahl als eine Zeilennummer-Referenz auf ein Statement. Deshalb muss einer vierstelligen Array-Ausprägung ein Schrägstrich (/) vorausgehen, um anzuzeigen, dass es sich um eine Array-Ausprägung handelt, zum Beispiel:

```
#ARRAY(/1000)
```

und nicht:

```
#ARRAY(1000)
```

weil das Letztere als eine Referenz auf die Sourcecode-Zeile 1000 interpretiert würde.

Wenn ein Index-Variablenname als eine Format/Längen-Angabe fehlinterpretiert werden könnte, muss ein Schrägstrich (/) benutzt werden, um anzuzeigen, dass ein Index angegeben wird. Wenn z.B. die Ausprägung eines Arrays durch den Wert der Variable `N7` definiert ist, muss die Ausprägung angegeben werden als:

```
#ARRAY (/N7)
```

und nicht:

```
#ARRAY (N7)
```

weil das Letztere als die Definition eines 7 Bytes umfassenden numerischen Feldes fehlinterpretiert würde.

Datenbank-Array referenzieren

Folgende Themen werden behandelt:

- Multiple Felder und Periodengruppen-Felder referenzieren
- Mit Konstanten definierte Arrays referenzieren
- Mit Variablen definierte Arrays referenzieren
- Mehrfach definierte Arrays referenzieren

Anmerkung:
Bevor Sie die folgenden Beispielprogramme ausführen, starten Sie bitte das Programm `INDEXTST` in der Library `SYSEXPG`, um einen Beispielsatz zu erstellen, der 10 verschiedene Sprachcodes verwendet.

Multiple Felder und Periodengruppen-Felder referenzieren

Ein multiples Feld oder ein Periodengruppen-Feld innerhalb eines Views bzw. einer DDM kann mittels verschiedener Index-Notationen definiert werden.

Zum Beispiel, der erste bis zehnte Wert und der Ite bis Ite+10 Wert desselben multiplen Feldes/Periodengruppen-Feldes eines Datenbanksatzes:

```
DEFINE DATA LOCAL
1 I (I2)
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 LANG (1:10)
  2 LANG (I:I+10)
END-DEFINE
```

oder:

```
RESET I (I2)
...
READ EMPLOYEES
OBTAIN LANG(1:10) LANG(I:I+10)
```

Anmerkungen:

1. Derselbe Index der unteren Grenze kann nur einmal pro Array benutzt werden (dies gilt für Konstanten-Indizes sowie für Variablen-Indizes).
2. Für eine Array-Definition unter Benutzung eines Variablen-Indexes muss die untere Grenze mittels der Variable selbst angegeben werden, und die obere Grenze muss mittels derselben Variable plus einer Konstante angegeben werden.

Mit Konstanten definierte Arrays referenzieren

Ein mit Konstanten definiertes Array kann entweder mittels Konstanten oder Variablen referenziert werden. Die obere Grenze des Array kann nicht überschritten werden. Die obere Grenze wird von Natural zur Kompilierungszeit geprüft, wenn eine Konstante benutzt wird.

Beispiel für den Reporting Mode:

```
** Example 'INDEX1R': Array definition with constants  (reporting mode)
************************************************************************
*
READ (1) EMPLOYEES WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  OBTAIN LANG (1:10)
  /*
  WRITE 'LANG(1:10):' LANG (1:10) //
  WRITE 'LANG(1)   :' LANG (1)    /  'LANG(5:9) :' LANG (5:9)
LOOP
*
END
```

Beispiel für den Structured Mode:

```
** Example 'INDEX1S': Array definition with constants (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 CITY
  2 LANG (1:10)
END-DEFINE
*
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  WRITE 'LANG(1:10):' LANG (1:10) //
  WRITE 'LANG(1)   :' LANG (1)    /  'LANG(5:9) :' LANG (5:9)
END-READ
END
```

Wenn ein multiples Feld oder ein Periodengruppen-Feld mehrmals mittels Konstanten definiert wird und mittels Variablen referenziert werden soll, wird die folgende Syntax benutzt.

Beispiel für den Reporting Mode:

```
** Example 'INDEX2R': Array definition with constants (reporting mode)
**                   (multiple definition of same database field)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 LANG (1:5)
  2 LANG (4:8)
END-DEFINE
*
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  DISPLAY 'NAME'         NAME
          'LANGUAGE/1:3' LANG (1.1:3)
          'LANGUAGE/6:8' LANG (4.3:5)
LOOP
*
END
```

Beispiel für den Structured Mode:

```
** Example 'INDEX2S': Array definition with constants (structured mode)
**                   (multiple definition of same database field)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 LANG (1:5)
  2 LANG (4:8)
END-DEFINE
*
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  DISPLAY 'NAME'         NAME
          'LANGUAGE/1:3' LANG (1.1:3)
          'LANGUAGE/6:8' LANG (4.3:5)
END-READ
*
END
```

Mit Variablen definierte Arrays referenzieren

Mit Variablen definierte multiple Felder oder Periodengruppen-Felder in Arrays müssen mittels derselben Variable referenziert werden.

Beispiel für den Reporting Mode:

```
** Example 'INDEX3R': Array definition with variables (reporting mode)
************************************************************************
RESET I (I2)
*
I := 1
READ (1) EMPLOYEES WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  OBTAIN LANG (I:I+10)
  /*
```

```
  WRITE 'LANG(I)      :' LANG (I) /
        'LANG(I+5:I+7):' LANG (I+5:I+7)
LOOP
*
END
```

Beispiel für den Structured Mode:

```
** Example 'INDEX3S': Array definition with variables (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 I (I2)
*
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 LANG (I:I+10)
END-DEFINE
*
I := 1
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  WRITE 'LANG(I)      :' LANG (I) /
        'LANG(I+5:I+7):' LANG (I+5:I+7)
END-READ
END
```

Wenn ein anderer Index benutzt werden soll, muss eine eindeutige Referenz auf die zuerst gefundene Definition des Arrays mit variablem Index erfolgen. Qualifizieren Sie dazu den Index-Ausdruck wie im Folgenden gezeigt.

Beispiel für den Reporting Mode:

```
** Example 'INDEX4R': Array definition with variables (reporting mode)
************************************************************************
RESET I (I2) J (I2)
*
I := 2
J := 3
*
READ (1) EMPLOYEES WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  OBTAIN LANG (I:I+10)
  /*
  WRITE 'LANG(I.J)  :' LANG (I.J) /
        'LANG(I.1:5):' LANG (I.1:5)
LOOP
*
END
```

Beispiel für den Structured Mode:

```
** Example 'INDEX4S': Array definition with variables (structured mode)
************************************************************************
DEFINE DATA LOCAL
1 I (I2)
1 J (I2)
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 LANG (I:I+10)
END-DEFINE
```

```
*
I := 2
J := 3
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  WRITE 'LANG(I.J)  :' LANG (I.J) /
        'LANG(I.1:5):' LANG (I.1:5)
END-READ
END
```

Der Ausdruck `I.` wird benutzt, um eine eindeutige Referenz auf die Array-Definition und "Positionen" zum ersten Wert innerhalb des Array-Lesebereichs (`LANG(I.1:5)`) zu erstellen.

Der aktuelle Inhalt von `I` zum Zeitpunkt des Datenbankzugriffs legt die Start-Ausprägung des Datenbank-Arrays fest.

Mehrfach definierte Arrays referenzieren

Für mehrfach definierte Arrays ist gewöhnlich eine Referenz mit Qualifizierung des Index-Ausdrucks erforderlich, um eine eindeutige Referenz auf den gewünschten Array-Bereich sicherzustellen.

Beispiel für den Reporting Mode:

```
** Example 'INDEX5R': Array definition with constants (reporting mode)
**                   (multiple definition of same database field)
************************************************************************
DEFINE DATA LOCAL                   /* For reporting mode programs
1 EMPLOY-VIEW VIEW OF EMPLOYEES     /* DEFINE DATA is recommended
  2 NAME                            /* to use multiple definitions
  2 CITY                            /* of same database field
  2 LANG (1:10)
  2 LANG (5:10)
*
1 I (I2)
1 J (I2)
END-DEFINE
*
I := 1
J := 2
*
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  WRITE 'LANG(1.1:10) :' LANG (1.1:10) /
        'LANG(1.I:I+2):' LANG (1.I:I+2) //
  WRITE 'LANG(5.1:5)  :' LANG (5.1:5)  /
        'LANG(5.J)    :' LANG (5.J)
LOOP
END
```

Beispiel für den Structured Mode:

```
** Example 'INDEX5S': Array definition with constants (structured mode)
**                   (multiple definition of same database field)
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 LANG (1:10)
  2 LANG (5:10)
*
1 I (I2)
```

```
1 J (I2)
END-DEFINE
*
*
I := 1
J := 2
*
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
  WRITE 'LANG(1.1:10) :' LANG (1.1:10) /
        'LANG(1.I:I+2):' LANG (1.I:I+2) //
  WRITE 'LANG(5.1:5)  :' LANG (5.1:5)  /
        'LANG(5.J)    :' LANG (5.J)
END-READ
END
```

Eine ähnliche Syntax wird auch benutzt, wenn multiple Felder oder Periodengruppen-Felder unter Benutzung von Index-Variablen definiert werden.

Beispiel für den Reporting Mode:

```
** Example 'INDEX6R': Array definition with variables (reporting mode)
**                   (multiple definition of same database field)
************************************************************************
DEFINE DATA LOCAL
1 I (I2) INIT <1>
1 J (I2) INIT <2>
1 N (I2) INIT <1>
1 EMPLOY-VIEW VIEW OF EMPLOYEES    /* For reporting mode programs
  2 NAME                           /* DEFINE DATA is recommended
  2 CITY                           /* to use multiple definitions
  2 LANG (I:I+10)                  /* of same database field
  2 LANG (J:J+5)
  2 LANG (4:5)
*
END-DEFINE
*
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
*
  WRITE 'LANG(I.I)    :' LANG (I.I) /
        'LANG(1.I:I+2):' LANG (I.I:I+10) //
*
  WRITE 'LANG(J.N)    :' LANG (J.N) /
        'LANG(J.2:4)  :' LANG (J.2:4) //
*
  WRITE 'LANG(4.N)    :' LANG (4.N) /
        'LANG(4.N:N+1):' LANG (4.N:N+1) /
LOOP
END
```

Beispiel für den Structured Mode:

```
** Example 'INDEX6S': Array definition with variables (structured mode)
**                   (multiple definition of same database field)
************************************************************************
DEFINE DATA LOCAL
1 I (I2) INIT <1>
1 J (I2) INIT <2>
1 N (I2) INIT <1>
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 LANG (I:I+10)
```

```
  2 LANG (J:J+5)
  2 LANG (4:5)
*
END-DEFINE
*
READ (1) EMPLOY-VIEW WITH NAME = 'WINTER' WHERE CITY = 'LONDON'
*
  WRITE 'LANG(I.I)    :' LANG (I.I) /
        'LANG(1.I:I+2):' LANG (I.I:I+10) //
*
  WRITE 'LANG(J.N)    :' LANG (J.N) /
        'LANG(J.2:4)  :' LANG (J.2:4) //
*
  WRITE 'LANG(4.N)    :' LANG (4.N) /
        'LANG(4.N:N+1):' LANG (4.N:N+1) /
END-READ
END
```

Internen Zähler für ein Datenbank-Array referenzieren — C*-Notation

Es ist manchmal erforderlich, ein multiples Feld und/oder eine Periodengruppe zu referenzieren, ohne zu wissen, wie viele Werte/Ausprägungen in einem gegebenen Datensatz vorhanden sind. Adabas unterhält einen internen Zähler der Anzahl für die Werte jedes einzelnen multiplen Feldes und die Anzahl der Ausprägungen jeder einzelnen Periodengruppe. Dieser Zähler kann unter Angabe von `C*` unmittelbar vor dem Feldnamen referenziert werden.

Anmerkung in Bezug auf andere Datenbanken als Adabas:

SQL	Bei SQL-Datenbanken kann die `C*` Notation nicht verwendet werden.
VSAM	Bei VSAM und DL/I-Datenbanken gibt die `C*` Notation nicht die Anzahl der Werte/Ausprägungen zurück, sondern die/den maximale/n Ausprägung/Wert, wie in der DDM definiert (`MAXOCC`).
DL/I	

Siehe auch das Zeilenkommando `.*` des Data-Area-Editors (in der *Editors*-Dokumentation).

Das/Die zum Deklarieren eines `C*`-Feldes zulässige explizite Format und Länge ist entweder

- Ganzzahl-Format Integer (I) mit einer Länge von 2 Bytes (I2) oder 4 Bytes (I4),
- numerisch (N) oder gepackt (P) mit einer ganzzahligen Ziffer (aber ohne Dezimalstellen), zum Beispiel (N3).

Wenn kein explizites Format und keine explizite Länge angegeben wird, ist Format/Länge N3 die Voreinstellung.

Beispiele:

C*LANG	gibt den Zähler der Anzahl der Werte für das multiple Feld LANG zurück.
C*INCOME	gibt den Zähler der Anzahl der Ausprägungen für die Periodengruppe INCOME zurück.
C*BONUS(1)	gibt den Zähler der Anzahl der Werte für das multiple Feld BONUS in der Periodengruppen-Ausprägung 1 zurück (wenn man davon ausgeht, dass BONUS ein multiples Fel.d innerhalb einer Periodengruppe ist.)

Beispielprogramm mit C*-Notation bei Variablen:

```
** Example 'CNOTX01': C* Notation
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 C*INCOME
  2 INCOME
    3 SALARY  (1:5)
    3 C*BONUS (1:2)
    3 BONUS   (1:2,1:2)
  2 C*LANG
  2 LANG      (1:2)
*
1 #I (N1)
END-DEFINE
*
LIMIT 2
READ EMPL-VIEW BY CITY
  /*
  WRITE NOTITLE 'NAME:' NAME /
       'NUMBER OF LANGUAGES SPOKEN:' C*LANG 5X
       'LANGUAGE 1:' LANG (1) 5X
       'LANGUAGE 2:' LANG (2)
  /*
  WRITE 'SALARY DATA:'
  FOR #I FROM 1 TO C*INCOME
    WRITE 'SALARY' #I SALARY (1.#I)
  END-FOR
  /*
  WRITE 'THIS YEAR BONUS:' C*BONUS(1)  BONUS (1,1) BONUS (1,2)
      / 'LAST YEAR BONUS:' C*BONUS(2)  BONUS (2,1) BONUS (2,2)
  SKIP 1
END-READ
END
```

Ausgabe des Programms CNOTX01:

```
NAME: SENKO
NUMBER OF LANGUAGES SPOKEN:    1     LANGUAGE 1: ENG     LANGUAGE 2:
SALARY DATA:
SALARY  1      36225
SALARY  2      29900
SALARY  3      28100
SALARY  4      26600
SALARY  5      25200
THIS YEAR BONUS:    0          0          0
LAST YEAR BONUS:    0          0          0

NAME: CANALE
```

```
NUMBER OF LANGUAGES SPOKEN:    2     LANGUAGE 1: FRE     LANGUAGE 2: ENG
SALARY DATA:
SALARY  1     202285
THIS YEAR BONUS:    1      23000          0
LAST YEAR BONUS:    0          0          0
```

C*-Notation bei multiplen Felder innerhalb von Periodengruppen

Für ein multiples Feld innerhalb einer Periodengruppe können Sie auch eine mit `C*`-Notation versehene Variable mit einer Indexbereichsangabe definieren.

Bei den folgenden Beispielen wird das multiple Feld `BONUS` benutzt, das Teil der Periodengruppe `INCOME` ist. Alle drei Beispiele liefern dasselbe Ergebnis.

Beispiel 1 — Reporting Mode:

```
** Example 'CNOTX02': C* Notation (multiple-value fields)
************************************************************************
*
LIMIT 2
READ EMPLOYEES BY CITY
  OBTAIN C*BONUS (1:3)
         BONUS   (1:3,1:3)
  /*
  DISPLAY NAME C*BONUS (1:3) BONUS (1:3,1:3)
LOOP
*
END
```

Beispiel 2 — Structured Mode:

```
** Example 'CNOTX03': C* Notation (multiple-value fields)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 INCOME    (1:3)
    3 C*BONUS
    3 BONUS   (1:3)
END-DEFINE
*
LIMIT 2
READ EMPL-VIEW BY CITY
  /*
  DISPLAY NAME C*BONUS (1:3) BONUS (1:3,1:3)
END-READ
*
END
```

Beispiel 3 — Structured Mode:

```
** Example 'CNOTX04': C* Notation (multiple-value fields)
************************************************************************
DEFINE DATA LOCAL
1 EMPL-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 CITY
  2 C*BONUS  (1:3)
  2 INCOME   (1:3)
```

```
    3 BONUS  (1:3)
END-DEFINE
*
LIMIT 2
READ EMPL-VIEW BY CITY
  /*
  DISPLAY NAME C*BONUS (*) BONUS (*,*)
END-READ
*
END
```

Warnung:
Da der Adabas-Formatpuffer keine Bereiche für Zählerfelder zulässt, werden sie als einzelne Felder generiert; deshalb kann ein `C*`-Indexbereich für ein großes Array zu einem Adabas Formatpuffer-Überlauf führen.

Datenstrukturen qualifizieren

Um ein Feld beim Referenzieren zu identifizieren, können Sie das Feld qualifizieren; d.h. vor dem Feldnamen geben Sie den Namen des Level-1-Datenelementes ein, in dem das Feld sich befindet, und einen Punkt.

Ist ein Feld nicht eindeutig über den Namen zu identifizieren (z.B. wenn derselbe Feldname in mehreren Gruppen/Views benutzt wird), müssen Sie es beim Referenzieren qualifizieren.

Die Kombination von Level-1-Datenelement und Feldnamen muss eindeutig sein.

Beispiel:

```
DEFINE DATA LOCAL
1 FULL-NAME
  2 LAST-NAME (A20)
  2 FIRST-NAME (A15)
1 OUTPUT-NAME
  2 LAST-NAME (A20)
  2 FIRST-NAME (A15)
END-DEFINE
...
MOVE FULL-NAME.LAST-NAME TO OUTPUT-NAME.LAST-NAME
...
```

Der Kennzeichner muss ein Level-1-Datenelement sein.

Beispiel:

```
DEFINE DATA LOCAL
1 GROUP1
  2 SUB-GROUP
    3 FIELD1 (A15)
    3 FIELD2 (A15)
END-DEFINE
...
MOVE 'ABC' TO GROUP1.FIELD1
...
```

Datenbankfeld qualifizieren:

Benutzen Sie denselben Namen für eine Benutzervariable und ein Datenbankfeld (was Sie nicht tun sollten), müssen Sie das Datenbankfeld qualifizieren, wenn Sie es referenzieren wollen.

Warnung:
Wenn Sie ein Datenbankfeld mit gleichem Namen wie eine Benutzervariable nicht qualifizieren, wird die Benutzervariable referenziert.

Beispiele für Benutzervariablen

```
DEFINE DATA LOCAL
1 #A1 (A10)       /* Alphanumeric, 10 positions.
1 #A2 (B4)        /* Binary, 4 positions.
1 #A3 (P4)        /* Packed numeric, 4 positions and 1 sign position.
1 #A4 (N7.2)      /* Unpacked numeric,
                  /* 7 positions before and 2 after decimal point.
1 #A5 (N7.)       /* Invalid definition!!!
1 #A6 (P7.2)      /* Packed numeric, 7 positions before and 2 after decimal point
                  /* and 1 sign position.
1 #INT1 (I1)      /* Integer, 1 byte.
1 #INT2 (I2)      /* Integer, 2 bytes.
1 #INT3 (I3)      /* Invalid definition!!!
1 #INT4 (I4)      /* Integer, 4 bytes.
1 #INT5 (I5)      /* Invalid definition!!!
1 #FLT4 (F4)      /* Floating point, 4 bytes.
1 #FLT8 (F8)      /* Floating point, 8 bytes.
1 #FLT2 (F2)      /* Invalid definition!!!
1 #DATE (D)       /* Date (internal format/length P6).
1 #TIME (T)       /* Time (internal format/length P12).
1 #SWITCH (L)     /* Logical, 1 byte (TRUE or FALSE).
                  /*
END-DEFINE
```