CALL

CALL [**INTERFACE4**] *operand1* [**USING**] [*operand2*] ... 128

Dieses Kapitel behandelt folgende Themen:

- Funktion
- Syntax-Beschreibung
- Return Code
- Registerbelegung
- Speicherplatzausrichtung
- Adabas-Aufrufe
- Direktes/Dynamisches Laden
- Linkage-Konventionen
- Aufrufen eines PL/I-Programms
- Aufruf eines C-Programms
- INTERFACE4

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

Verwandte Statements: CALL FILE | CALL LOOP | CALLNAT | DEFINE SUBROUTINE | ESCAPE | FETCH | PERFORM

Gehört zur Funktionsgruppe: *Aufrufen von Programmen und Unterprogrammen*

Funktion

Mit dem CALL-Statement können Sie von einem Natural-Programm aus ein anderes, in einer anderen Standard-Programmiersprache geschriebenes Programm aufrufen, wobei im Anschluss daran die Verarbeitung des Natural-Programms mit dem nächsten Statement nach dem CALL-Statement fortgesetzt wird.

Das aufgerufene Program kann in einer beliebigen anderen Programmiersprache, die eine Standard-CALL-Schnittstelle unterstützt, geschrieben sein. Mehrere CALL-Statements können verwendet werden, um ein Programm mehrmals oder mehrere Programme aufzurufen.

Ein CALL-Statement kann auch in einem Programm, das unter Kontrolle eines TP-Monitors ausgeführt werden soll, angegeben werden, vorausgesetzt der TP-Monitor unterstützt eine CALL-Schnittstelle.

Syntax-Beschreibung

Operanden-Definitionstabelle:

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

Syntax-Element-Beschreibung:

INTERFACE4	Das optionale Schlüsselwort `INTERFACE4` gibt den Typ der Schnittstelle an, die für den Aufruf des externen Programms verwendet wird. Siehe den Abschnitt *INTERFACE4* weiter unten.
operand1	**Programmname:** Der Name des aufgerufenen Programms (*operand1*) kann entweder als Konstante angegeben werden oder — falls je nach Programmlogik verschiedene Programme aufgerufen werden sollen — als alphanumerische Variable mit Länge 1 bis 8. Ein Programmname muss linksbündig in der Variablen stehen.
[USING] ***operand2***	**Parameter:** Das `CALL`-Statement kann bis zu 128 Parameter (*operand2*) enthalten, es sei denn, die `INTERFACE4`-Option wird verwendet. In diesem Fall wird die Anzahl der Parameter durch die Größe des katalogisierten Objekts begrenzt. Abhängig von allen anderen Statements im Natural-Objekt dürfen bis zu 16370 Parameter benutzt werden. Hierbei gelten die Standard-Linkage-Registerkonventionen. Für jedes angegebene Parameterfeld wird in der Parameterliste eine Adresse übergeben. Wird ein Gruppenname verwendet, so wird die Gruppe in einzelne Felder umgesetzt, d.h. der Benutzer muss das erste Feld der Gruppe angeben, falls er die Anfangsadresse einer Gruppe spezifizieren will. Die interne Darstellung der positiven Vorzeichen gepackter Zahlen ändert sich auf den vom `PSIGNF`-Parameter des `NTCMPO`-Makros (Kompilieroptionen) angegebenen Wert, *bevor* die Kontrolle an das externe Programm übergeben wird.

Return Code

Um den Condition Code eines aufgerufenen Programms (Inhalt von Register 15 beim Rücksprung zum Natural-Programm) zu erhalten, können Sie die Natural-Systemfunktion `RET` verwenden.

Beispiel:

```
...
RESET #RETURN(B4)
CALL 'PROG1'
IF RET ('PROG1') > #RETURN
  WRITE 'ERROR OCCURRED IN PROGRAM1'
END-IF
...
```

Registerbelegung

Register	Inhalt
R1	Adresse der Parameteradressenliste.
R2	Adresse der Feld-(Parameter-)Beschreibungsliste. Die Feldbeschreibungsliste enthält Informationen über die ersten 128 in der Parameterliste übergebenen Felder. Jede Beschreibung ist ein 4 Byte langer Eintrag, der sich wie folgt zusammensetzt: • 1. Byte: Variablentyp (A, B, ...) • Überschreitet eine Variable vom Typ A die Länge von 32767 Bytes, wird sie als Typ Y übergeben. • Überschreitet eine Variable vom Typ B die Länge von 32767 Bytes, wird sie als Typ X übergeben. Die Typen X und Y wurden zur Unterstützung langer alphanumerischer und binärer Vairablen für die standardmäßige CALL-Schnittstelle eingeführt. Falls das Feld vom Typ N oder P ist: • 2. Byte: Anzahl der Stellen insgesamt. • 3. Byte: Anzahl der Stellen vor dem Komma (Dezimalpunkt). • 4. Byte: Anzahl der Stellen nach dem Komma (Dezimalpunkt). Falls das Feld vom Typ X oder Y ist: • 2. Byte: ungenutzt. • 3.–4. Byte: enthält Null. • Länge des Feldes wird über R4 übergeben. Bei allen anderen Feldtypen: • 2. Byte: ungenutzt. • 3.–4. Byte: Länge des Feldes.

Register	Inhalt
R3	Adresse der Feldlängen-Liste. Diese Liste enthält die Längen der in der Parameterliste übergebenen Felder. Bei einem Array ergibt sich die Länge aus der Summe der Längen der einzelnen Ausprägungen.
R4	Nur bei Typ X und Y: • Ein vier Byte langer Eintrag für jede Variable des Typs A oder B, die die Größe von 32767 Bytes überschreitet.
R13	Adresse des 18-Wort-Speicherbereiches.
R14	Rücksprungadresse.
R15	Eingangsadresse/Return Code.

Speicherplatzausrichtung

Siehe Abschnitt *Speicherplatzausrichtung* im *Leitfaden zur Programmierung*.

Adabas-Aufrufe

Ein aufgerufenes Programm darf einen Adabas-Aufruf beinhalten. Das aufgerufene Programm darf kein Adabas-Open- oder -Close-Kommando absetzen. Adabas öffnet alle angesprochenen Dateien.

Soll Adabas-EXU-Modus (EXclusive Update) verwendet werden, muss zum Öffnen aller angesprochenen Dateien der Natural-Profilparameter `OPRB` (Öffnen und Schließen der Datenbank) benutzt werden; bevor Sie den EXU-Update-Modus einsetzen, sollten Sie in jedem Fall Ihren Natural-Administrator konsultieren.

Wenn ein aufgerufenes Programm Adabas-Kommandos absetzt, die eine Transaktion beginnen oder beenden, kann Natural nicht die Änderung des Transaktions-Status erkennen.

Aufrufe an Adabas müssen den Aufruf-Konventionen für die Adabas-Anwendungsprogrammierschnittstelle (API) für den/das entsprechende/n TP-Monitor oder Betriebssystem entsprechen. Dies gilt auch, wenn Natural als Server agiert, z.B. unter z/OS oder SMARTS.

Direktes/Dynamisches Laden

Das aufgerufene Programm kann entweder direkt an den Natural-Nukleus gelinkt werden (indem es mit dem `CSTATIC`-Parameter im Natural-Parametermodul angegeben wird; vgl. *Operations*-Dokumentation), oder es kann dynamisch geladen werden, wenn es zum erstenmal aufgerufen wird.

Soll es dynamisch geladen werden, so muss die Lademodul-Library, die das aufgerufene Programm enthält, mit der Natural-Lade-Library verkettet werden, und zwar entweder in der Natural-Ausführungs-JCL oder in der entsprechenden Programm-Library des TP-Monitors. Weitere Einzelheiten erfragen Sie bitte bei Ihrem Natural-Administrator.

Beispiel:

Das Beispiel auf der nächsten Seite zeigt ein Natural-Programm, welches das COBOL-Programm TABSUB aufruft, und zwar zu dem Zweck, Ländercodes (COUNTRY-CODE) in die entsprechenden Ländernamen (COUNTRY-NAME) umzusetzen. Das Natural-Programm übergibt zwei Parameter an das COBOL-Programm:

- der erste Parameter ist der Ländercode, so wie er von der Datenbank gelesen wird;
- der zweite Parameter dient dazu, den Ländernamen zurückzugeben.

Aufrufendes Natural-Programm:

```
** Example 'CALEX1': CALL PROGRAM 'TABSUB'
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 BIRTH
  2 COUNTRY
*
1 #COUNTRY      (A3)
1 #COUNTRY-NAME (A15)
1 #FIND-FROM    (D)
1 #FIND-TO      (D)
END-DEFINE
*
MOVE EDITED '19550701' TO #FIND-FROM (EM=YYYYMMDD)
MOVE EDITED '19550731' TO #FIND-TO   (EM=YYYYMMDD)
*
FIND EMPLOY-VIEW WITH BIRTH = #FIND-FROM THRU #FIND-TO
  MOVE COUNTRY TO #COUNTRY
  /*
  CALL 'TABSUB' #COUNTRY #COUNTRY-NAME
  /*
  DISPLAY NAME BIRTH (EM=YYYY-MM-DD) #COUNTRY-NAME
END-FIND
END
```

Aufgerufenes COBOL-Programm TABSUB:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TABSUB.
REMARKS. THIS PROGRAM PROVIDES THE COUNTRY NAME
        FOR A GIVEN COUNTRY CODE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
LINKAGE SECTION.
01 COUNTRY-CODE  PIC X(3).
01 COUNTRY-NAME  PIC X(15).
PROCEDURE DIVISION USING COUNTRY-CODE COUNTRY-NAME.
P-CONVERT.
   MOVE SPACES TO COUNTRY-NAME.
   IF COUNTRY-CODE = 'BLG' MOVE 'BELGIUM' TO COUNTRY-NAME.
   IF COUNTRY-CODE = 'DEN' MOVE 'DENMARK' TO COUNTRY-NAME.
   IF COUNTRY-CODE = 'FRA' MOVE 'FRANCE' TO COUNTRY-NAME.
   IF COUNTRY-CODE = 'GER' MOVE 'GERMANY' TO COUNTRY-NAME.
   IF COUNTRY-CODE = 'HOL' MOVE 'HOLLAND' TO COUNTRY-NAME.
   IF COUNTRY-CODE = 'ITA' MOVE 'ITALY' TO COUNTRY-NAME.
```

```
   IF COUNTRY-CODE = 'SPA' MOVE 'SPAIN' TO COUNTRY-NAME.
   IF COUNTRY-CODE = 'UK'  MOVE 'UNITED KINGDOM' TO COUNTRY-NAME.
P-RETURN.
GOBACK.
```

Linkage-Konventionen

Im Batch-Betrieb wird Standard-Linkage-Register-Notation verwendet. Für jeden TP-Monitor gelten hierbei andere Konventionen. Diese Konventionen müssen unbedingt befolgt werden, da es sonst zu unvorhersehbaren Ergebnissen kommen kann.

Die für die einzelnen unterstützten TP-Monitore gültigen Konventionen sind im folgenden beschrieben:

- CALL unter Com-plete
- CALL unter CICS

CALL unter Com-plete

Das aufgerufene Programm muss sich in der Com-plete-Online-Lade-Library befinden, damit Com-plete es dynamisch laden kann. Zum Katalogisieren des Programms kann die Com-plete-Utility `ULIB` verwendet werden.

CALL unter CICS

Das aufgerufene Programm muss sich entweder in der Library DFHRPL befinden oder in der Lademodul-Library, die mit der CICS-Library verkettet ist. Damit CICS das Programm finden und laden kann, muss das Programm außerdem in der aktiven PPT eingetragen sein.

Die Linkage-Konvention übergibt die Parameteradressenliste sowie die Feldbeschreibungsliste in den ersten Vollwörtern der TWA und der COMMAREA.

Der Parameter `FLDLEN` im Parametermodul `NCIPARM` bestimmt, ob die Feldlängenliste auch übergeben wird (standardmäßig wird sie nicht übergeben). An der Länge der COMMAREA (0, 8, 12 oder 16) lässt sich erkennen, wieviele Listenadressen (0, 2, 3 oder 4) übergeben werden. Die letzte Listenadresse wird dadurch angezeigt, dass das linke Bit gesetzt ist. Es ist Sache des Benutzers, die Adressierbarkeit der TWA bzw. der COMMAREA zu gewährleisten, falls das aufgerufene Programm für Natural nicht als statisches oder direkt gelinktes Programm definiert ist; in diesem Fall werden die Adresse der Parameterliste über Register 1, die Adresse der Feldbeschreibungsliste über Register 2, die Adresse der Feldlängenliste in Register 3 und die Adresse der Liste großer Feldlängen in Register 4 übergeben.

Die Parameter `FLDLEN` und `COMACAL` im `NCIPARM`-Parametermodul kontrollieren den Inhalt der COMMAREA.

Wenn Sie statt der Adresse der Parameteradressenliste die Parameterwerte selbst in der COMMAREA übergeben möchten, führen Sie vor dem Aufruf das Terminalkommando `%P=C` aus.

Wenn ein Natural-Programm ein Nicht-Natural-Programm aufruft und das aufgerufene Programm im Dialog einen Terminal-I/O absetzt, ist der Natural-Thread normalerweise solange blockiert, bis der Benutzer eine Eingabe gemacht hat. Um zu vermeiden, dass der Thread blockiert ist, können Sie das Terminalkommando `%P=V` verwenden.

Wenn ein Natural-Programm unter CICS ein Nicht-Natural-Programm aufruft, geschieht der Aufruf normalerweise über eine EXEC CICS LINK-Anforderung. Wenn für den Aufruf stattdessen Standard Linkage verwendet werden soll, geben Sie das Terminalkommando %P=S ein. (Voraussetzung ist, dass das aufgerufene Programm den Standard-Linkage-Konventionen mit Standard-Registerverwendung entspricht.)

In 31-Bit-Modus-Umgebungen gilt folgendes: wenn ein mit AMODE=24 gelinktes Programm aufgerufen wird und die Threads liegen über 16 MB, wird automatisch ein Wertaufruf (Call by Value) durchgeführt, d.h. die angegebenen Parameter, die an das aufgerufene Programm übergeben werden sollen, werden in den Bereich unterhalb 16 MB kopiert.

Return Codes unter CICS

CICS selbst unterstützt zwar keine Return Codes für einen Aufruf mit CICS-Konventionen (EXEC CICS LINK), außer beim Aufruf von C/C++-Programmen, wobei von der exit()-Funktion oder dem return()-Statement übergebene Werte im EIBRESP2-Feld gespeichert werden. Aber das Natural/CICS-Interface unterstützt Return Codes für das CALL-Statement: Wenn die Kontrolle vom aufgerufenen Programm zurückgegeben wird, überprüft Natural zuerst das EIBRESP2-Feld auf einen Return Code ungleich Null. Dann überprüft Natural, ob sich das erste Vollwort der COMMAREA geändert hat. Ist dies der Fall, wird dessen neuer Inhalt als Return Code genommen. Hat es sich nicht geändert, wird das erste Vollwort der TWA geprüft und dessen neuer Inhalt als Return Code genommen. Hat sich keins der beiden Vollwörter geändert, ist der Return Code 0.

Anmerkung:
Wenn die Parameterwerte in der COMMAREA übergeben werden (%P=C), wird nur das Feld EIBRESP2 auf einen Returncode überprüft, d.h. bei Nicht-C/C++-Programmen ist der Return Code immer 0.

Beispiel unter CICS:

```
IDENTIFICATION DIVISION.
PROGRAM-ID. TABSUB.
REMARKS. THIS PROGRAM PERFORMS A TABLE LOOK-UP AND
         RETURNS A TEXT MESSAGE.
ENVIRONMENT DIVISION.
DATA DIVISION.
WORKING-STORAGE SECTION.
01 MSG-TABLE.
   03  FILLER       PIC X(15) VALUE 'MESSAGE1       '.
   03  FILLER       PIC X(15) VALUE 'MESSAGE2       '.
   03  FILLER       PIC X(15) VALUE 'MESSAGE3       '.
   03  FILLER       PIC X(15) VALUE 'MESSAGE4       '.
   03  FILLER       PIC X(15) VALUE 'MESSAGE5       '.
   03  FILLER       PIC X(15) VALUE 'MESSAGE6       '.
   03  FILLER       PIC X(15) VALUE 'MESSAGE7       '.
01 TAB REDEFINES MSG-TABLE.
   03  MESSAGE OCCURS 7 TIMES PIC X(15).
LINKAGE SECTION.
01 TWA-DATA.
   03 PARM-POINTER USAGE IS POINTER.
01 PARM-LIST.
   03 DATA-LOC-IN  USAGE IS POINTER.
   03 DATA-LOC-OUT USAGE IS POINTER.
01 INPUT-DATA.
   03  INPUT-NUMBER      PIC 99.
01 OUTPUT-DATA.
   03  OUTPUT-MESSAGE    PIC X(15).
PROCEDURE DIVISION.
```

```
100-INIT.
    EXEC CICS ADDRESS TWA(ADDRESS OF TWA-DATA) END-EXEC.
    SET ADDRESS OF PARM-LIST   TO PARM-POINTER.
    SET ADDRESS OF INPUT-DATA  TO DATA-LOCIN.
    SET ADDRESS OF OUTPUT-DATA TO DATA-LOC-OUT.
200-PROCESS.
    MOVE MESSAGE (INPUT-NUMBER) TO OUTPUT-MESSAGE.
300-RETURN.
    EXEC CICS RETURN END-EXEC.
400-DUMMY.
    GO-BACK.
```

Aufrufen eines PL/I-Programms

Ist das aufgerufene Programm in PL/I geschrieben, erfordert dies zusätzlich folgendes:

- Da die Parameterliste eine Standardliste und keine von einem anderen PL/I-Programm übergebene Argumentliste ist, zeigen die übergebenen Adressen nicht auf einen `LOCAL DESCRIPTOR`. Dieses Problem löst man, indem man die Parameterfelder als arithmetische Variablen definiert. Dies bewirkt, dass PL/I die Parameterliste als Adressen von Daten und nicht als Adressen von `LOCAL-DESCRIPTOR`-Kontrollblöcken behandelt.

Es wird empfohlen, die Parameterfelder folgendem Beispiel entsprechend zu definieren:

```
PLIPROG: PROC(INPUT_PARM_1, INPUT_PARM_2) OPTIONS(MAIN);
      DECLARE (INPUT_PARM_1, INPUT_PARM_2) FIXED;
      PTR_PARM_1 = ADDR(INPUT_PARM_1);
      PTR_PARM_2 = ADDR(INPUT_PARM_2);
      DECLARE FIRST_PARM        PIC '99'   BASED (PTR_PARM_1);
      DECLARE SECOND_PARM       CHAR(12)   BASED (PTR_PARM_2);
```

Jeder Parameter der Input-Liste sollte als eindeutiges Element behandelt werden. Die Anzahl der Eingabeparameter sollte mit der Zahl der vom Natural-Programm übergebenen genau übereinstimmen. Die Eingabeparameter und ihre Attribute müssen den Natural-Definitionen entsprechen, andernfalls kann es zu unvorhersehbaren Ergebnissen kommen. Weitere Informationen zur Übergabe von Parametern an PL/I-Programme finden Sie in der entsprechenden IBMPL/I-Dokumentation.

Siehe auch die folgenden Beispiele:

- Beispiel für den Aufruf eines PL/I-Programms
- Beispiel für den Aufruf eines PL/I-Programms, das unter CICS abläuft

Beispiel für den Aufruf eines PL/I-Programms

```
** Example 'CALEX2': CALL PROGRAM 'NATPLI'
************************************************************************
DEFINE DATA LOCAL
1 EMPLOY-VIEW VIEW OF EMPLOYEES
  2 NAME
  2 AREA-CODE
  2 REDEFINE AREA-CODE
    3 #AC         (N1)
*
1 #INPUT-NUMBER   (N2)
1 #OUTPUT-COMMENT (A15)
END-DEFINE
```

```
*
READ EMPLOY-VIEW IN LOGICAL SEQUENCE BY NAME
                 STARTING FROM 'WAGNER'
  MOVE ' ' TO #OUTPUT-COMMENT
  MOVE #AC TO #INPUT-NUMBER
  /*
  CALL 'NATPLI' #INPUT-NUMBER #OUTPUT-COMMENT
  /*
END-READ
*
END
```

Aufgerufenes PL/I-Programm `NATPLI`:

```
NATPLI:  PROC(PARM_COUNT, PARM_COMMENT) OPTIONS(MAIN);
    /*                                                */
    /* THIS PROGRAM ACCEPTS AN INPUT NUMBER           */
    /* AND TRANSLATES IT TO AN OUTPUT CHARACTER       */
    /* STRING FOR PLACEMENT ON THE FINAL              */
    /* NATURAL REPORT                                 */
    /*                                                */
    /*                                                */
   DECLARE PARM_COUNT, PARM_COMMENT  FIXED;
   DECLARE ADDR BUILTIN;
   COUNT_PTR = ADDR(PARM_COUNT);
   COMMENT_PTR = ADDR(PARM_COMMENT);
   DECLARE  INPUT_NUMBER    PIC '99' BASED (COUNT_PTR);
   DECLARE  OUTPUT_COMMENT  CHAR(15) BASED (COMMENT_PTR);
   DECLARE COMMENT_TABLE(9) CHAR(15) STATIC INITIAL
      ('COMMENT1   ',
       'COMMENT2   ',
       'COMMENT3   ',
       'COMMENT4   ',
       'COMMENT5   ',
       'COMMENT6   ',
       'COMMENT7   ',
       'COMMENT8   ',
       'COMMENT9   ');
    OUTPUT_COMMENT = COMMENT_TABLE(INPUT_NUMBER);
    RETURN;
END NATPLI;
```

Beispiel für den Aufruf eines PL/I-Programms, das unter CICS abläuft

```
** Example 'CALEX3': CALL PROGRAM 'CICSP'
************************************************************************
DEFINE DATA LOCAL
1 #MESSAGE (A10) INIT <' '>
END-DEFINE
*
CALL 'CICSP' #MESSAGE
DISPLAY #MESSAGE
*
END
```

Aufgerufenes PL/I-Programm `CICSP`:

```
CICSP: PROCEDURE OPTIONS (MAIN REENTRANT);
       DCL  1        TWA_ADDRESS    BASED(TWA_POINTER);
            2        LIST_ADDRESS   POINTER;
       DCL  1 PTR_TO_LIST           BASED(LIST_ADDRESS);
            2 PARM_01               POINTER;
       DCL MESSAGE CHAR(10) BASED(PARM_01);
       EXEC CICS ADDRESS TWA(TWA_POINTER);
       MESSAGE='SUCCESS'; EXEC CICS RETURN; END CICSP;
```

Aufruf eines C-Programms

Bevor Sie ein C-Programm benutzen können, müssen Sie es zuerst kompilieren und linken.

- Zum Erstellen des ausführbaren Moduls benutzen Sie z.B. den C Compiler von IBM. Da der C Compiler von IBM LE-Code erzeugt, ist das Muster nur in einer LE-Umgebung ausführbar. Um LE-Programme auszuführen, muss das Natural-Frontend mit LE-Funktionalität installiert werden.
- Wenn Sie einen anderen C Compiler wie z.B. Dignus oder SASC benutzen möchten, ist es erforderlich, dass Sie ein Modul erstellen, das von einer Nicht-C-Umgebung aufrufbar ist. Weitere Informationen entnehmen Sie der betreffenden Compiler-Dokumentation.
- Die Include-Datei `NATUSER` muss im C-Programm mit enthalten sein.

Für `INTERFACE4` geschriebene C-Programme können sowohl auf Großrechner-Systemen als auch auf UNIX-, OpenVMS- oder Windows-Systemen verwendet werden. Dagegen sind für die standardmäßige CALL-Schnittstelle geschriebene C-Programme plattformabhängig.

Wenn das C-Programm über `CALL INTERFACE4` aufgerufen werden soll oder wenn ein Natural-Subprogramm vom C-Programm aufgerufen wird, muss `NATXCAL4` an das ausführbare Modul gelinkt werden. Benutzen Sie eine der `INTERFACE4`-Rückruffunktionen, um die Parameter-Beschreibung und Parameter-Werte einzulesen. Die Rückruf-Funktionen sind im folgenden beschrieben.

Benutzen Sie die Funktion `ncxr_if4_callnat`, um ein Natural-Subprogramm vom C-Programm aus auszuführen.

Prototyp:

```
int ncxr_if4_callnat ( char *natpgm, int parmnum, struct parameter_description *descr );
```

Parameter-Beschreibung:

natpgm	Name des aufzurufenden Natural-Subprogramms.	
parmnum	Anzahl der an das Subprogramm zu übergebenden Parameter-Felder.	
descr	Addresse einer `struct parameter_description`. Siehe *Operanden-Struktur für Interface4*.	
return	Rückgabewert:	Informationen:
	`0`	OK Tritt ein Natural-Fehler auf, während das Subprogramm ausgeführt wird, werden Informationen über diesen Fehler in der Variable `natpgm` in der Form `*NAT`*nnnn* zurückgegeben, wobei *nnnn* die betreffende Natural-Fehlernummer ist.
	`-1`	Ungültige Parameter-Nummer.
	`-2`	Interner Fehler.

Siehe auch die folgenden Beispiele:

- Beispiel zum Aufruf eines C-Programms über einen Standard-CALL
- Beispiel zum Aufruf eines C-Programms über CALL INTERFACE4

Beispiel zum Aufruf eines C-Programms über einen Standard-CALL

```
** Example 'CALEX4': CALL PROGRAM 'ADD'
************************************************************************
DEFINE DATA LOCAL
1 #OP1  (I4)
1 #OP2  (I4)
1 #SUM  (I4)
END-DEFINE
*
CALL 'ADD' #OP1 #OP2 #SUM
DISPLAY #SUM
*
END
```

Aufgerufenes C-Programm `ADD`:

```
/*
**  Example C Program ADD.c
*/
NATFCT ADD (int *op1, int *op2, int *sum)
{
*sum = *op1 + *op2;    /* add opperands */

return 0;              /* return successfully */
} /* ADD */
```

Beispiel zum Aufruf eines C-Programms über CALL INTERFACE4

```
** Example 'CALEX5': CALL PROGRAM 'ADD4'
************************************************************************
DEFINE DATA LOCAL
1 #OP1  (I4)
1 #OP2  (I4)
```

```
1 #SUM  (I4)
END-DEFINE
*
CALL INTERFACE4 'ADD4' #OP1 #OP2 #SUM
DISPLAY #SUM
*
END
```

Aufgerufenes C-Programm ADD4:

```
NATFCT ADD4 NATARGDEF(numparm, parmhandle, parmdec)
{
NATTYP_I4 op1, op2, sum;              /* local integers */
int i;                                /* loop counter */
struct parameter_description desc;
int rc;                               /* return code access Funktions */

/*
** test number of arguments
*/
if (numparm != 3) return 1;

/*
** test types of arguments
*/
for (i = 0; i < (int) numparm; i++)
{
       rc = ncxr_get_parm_info( i, parmhandle, &desc );
      if ( rc ) return rc;

      if ( desc.format != 'I' || desc.length != sizeof(NATTYP_I4) || desc.dimensions != 0 )
      {
              return 2;          /* invalid parameter */
      }
}

/*
** get arguments
*/
rc = ncxr_get_parm( 0, parmhandle, sizeof op1, (void *)&op1 );
if ( rc ) return rc;

rc = ncxr_get_parm( 1, parmhandle, sizeof op2, (void *)&op2 );
if ( rc ) return rc;

/*
** perform the addition
*/
sum = op1 + op2;

/*
** move the result back to operand 3
*/
rc = ncxr_put_parm( 2, parmhandle, sizeof sum, (void *)&sum );
if ( rc ) return rc;

/*
** all ok, return success to the caller
*/
return 0;
} /* ADD4 */
```

INTERFACE4

Das Schlüsselwort `INTERFACE4` gibt den Typ der Schnittstelle an, die zum Aufruf des externen Programms verwendet wird. Dieses Schlüsselwort ist optional. Wenn dieses Schlüsselwort angegeben wird, wird die als `INTERFACE4` definierte Schnittstelle zum Aufruf des externen Programms verwendet.

Folgende Themen werden behandelt:

- Unterschiede zwischen CALL-Statement mit und ohne INTERFACE4
- INTERFACE4 — Externe 3GL-Programmierschnittstelle
- Operanden-Struktur für INTERFACE4
- INTERFACE4 — Parameter-Zugriff
- Exportierte Funktionen

Unterschiede zwischen CALL-Statement mit und ohne INTERFACE4

Die folgende Tabelle zeigt die Unterschiede zwischen dem mit `INTERFACE4` benutzten `CALL`-Statement und dem ohne `INTERFACE4` benutzten.

	CALL-Statement ohne Schlüsselwort INTERFACE4	**CALL-Statement mit Schlüsselwort INTERFACE4**
Anzahl der möglichen Parameter	128	16370 oder weniger
Maximale Länge eines Parameters	keine Beschränkung	1 GB
Array-Informationen einlesen	nein	ja
Unterstützung großer und dynamischer Operanden	voller Lesezugriff, Schreiben ohne Ändern der Größe des Operanden	ja
Parameter-Zugriff über API	direkt	über API

Die maximale Anzahl der Parameter wird durch die maximale Größe des generierten Programms (GP) und durch die maximale Größe eines Statements beschränkt. 16370 Parameter sind möglich, wenn das Programm nur das `CALL`-Statement enthält. Die maximale Anzahl ist niedriger, wenn andere Statements benutzt werden.

INTERFACE4 — Externe 3GL-Programmierschnittstelle

Die Schnittstelle des externen 3GL-Programms wird wie folgt definiert, wenn `INTERFACE4` im Natural-`CALL`-Statement angegeben wird:

```
NATFCT functionname (numparm, parmhandle, traditional)
```

USR_WORD	numparm;	16 Bit umfassender Kurzwert ohne Vorzeichen, der die Gesamtzahl der übertragenen Operanden (*operand2*) enthält
void	*parmhandle;	Adresse der Parameterübergabe-Struktur.
void	*traditional;	Schnittstellen-Typ prüfen (wenn es keine NULL-Adresse ist, handelt es sich um die herkömmliche `CALL`-Schnittstelle).

Operanden-Struktur für INTERFACE4

Die Operanden-Struktur von `INTERFACE4` wird als `parameter_description` bezeichnet und ist wie folgt definiert. Die Struktur wird mit der Header-Datei *natuser.h* ausgeliefert.

struct parameter_description		
`void *`	`address`	Adresse der Parameterdaten, nicht ausgerichtet, `realloc()` und `free()` sind nicht zulässig.
`int`	`format`	Felddatentyp: `NCXR_TYPE_ALPHA`, usw. (*natuser.h*).
`int`	`length`	Länge vor Dezimalpunkt (wenn zutreffend).
`int`	`precision`	Länge hinter Dezimalpunkt (wenn zutreffend).
`int`	`byte_length`	Länge des Feldes in Bytes; `int` Dimensionszahl (`0` bis `IF4_MAX_DIM`)
`int`	`dimensions`	Anzahl Dimensionen (`0` bis `IF4_MAX_DIM`).
`int`	`length_all`	Gesamtlänge des Arrays in Bytes.

`int`	`flags`	Mehrere Flag-Bits, durch `OR` bitweise miteinander kombiniert, Bedeutung:	
		`IF4_FLG_PROTECTED`	Parameter ist schreibgeschützt.
		`IF4_FLG_DYNAMIC`	Parameter ist dynamische Variable.
		`IF4_FLG_NOT_CONTIGUOUS`	Array-Elemente berühren sich nicht (es steht ein Leerzeichen zwischen ihnen).
		`IF4_FLG_AIV`	Der Parameter ist eine anwendungsunabhängige Variable.
		`IF4_FLG_DYNVAR`	Parameter ist dynamische Variable.
		`IF4_FLG_XARRAY`	Parameter ist ein X-Array.
		`IF4_FLG_LBVAR_0`	Untere Grenze der Dimension 0 ist variabel.
		`IF4_FLG_UBVAR_0`	Obere Grenze der Dimension 0 ist variabel.
		`IF4_FLG_LBVAR_1`	Untere Grenze der Dimension 1 ist variabel.
		`IF4_FLG_UBVAR_1`	Obere Grenze der Dimension 1 ist variabel.
		`IF4_FLG_LBVAR_2`	Untere Grenze der Dimension 2 ist variabel.
		`IF4_FLG_UBVAR_2`	Obere Grenze der Dimension 2 ist variabel.
`int`	`occurrences[IF4_MAX_DIM]`	Array-Ausprägungen in jeder Dimension.	
`int`	`indexfactors[IF4_MAX_DIM]`	Array-Index-Faktoren für jede Dimension.	
`void *`	`dynp`	Reserviert für interne Zwecke.	
`void *`	`pops`	Reserviert für interne Zwecke.	

Das Adress-Element ist Null für Arrays dynamischer Variablen und für X-Arrays. In diesen Fällen kann auf die Array-Daten nicht als Ganzes zugegriffen werden, sondern es muss über die unten beschriebenen Parameterzugriffsfunktionen auf sie zugegriffen werden.

Für Arrays mit festen Grenzen von Variablen fester Länge kann auf den Array-Inhalt direkt über das Adress-Element zugegriffen werden. In diesen Fällen errechnet sich die Adresse eines Array-Elements (i, j, k) wie folgt (besonders, wenn die Array-Elemente sich nicht berühren):

```
elementaddress = address + i * indexfactors[0] + j * indexfactors[1] + k * indexfactors[2]
```

Wenn das Array weniger als 3 Dimensionen hat, entfallen die letzten Ausdrücke.

INTERFACE4 — Parameter-Zugriff

Eine Reihe von Funktionen steht für den Zugriff auf die Parameter zur Verfügung. Der Ablauf der Verarbeitung ist wie folgt.

- Das 3GL-Programm wird über das `CALL`-Statement mit der Option `INTERFACE4` aufgerufen, und die Parameter werden an das 3GL-Programm wie oben beschrieben übergeben.
- Das 3GL-Programm kann jetzt die exportierten Funktionen von Natural verwenden, um entweder die Parameterdaten selbst oder Informationen über die Parameter, wie Format, Länge, Array-Informationen usw. einzulesen.
- Die exportierten Funktionen dienen auch dazu, Parameterdaten zurückzugeben.

Es gibt außerdem Funktionen zum Erstellen und Initialisieren eines neuen Parameter-Sets, um arbiträre Subprogramme von einem 3GL-Programm aus aufzurufen. Mit dieser Technik ist der Zugriff auf Parameter gewährleistet, um zu verhindern, dass das 3GL-Programm den Speicher überschreibt. Natural-Daten sind sicher — ein Überschreiben des Speichers im Bereich der Daten des 3GL-Programms ist noch möglich.

Exportierte Funktionen

Folgende Themen werden behandelt:

- Parameter-Informationen holen
- Parameterdaten holen
- Operanden-Daten zurückschreiben
- Parameter-Set erstellen, initialisieren und löschen
- Parameter-Set erstellen
- Parameter-Set löschen
- Skalar eines statischen Datentyps initialisieren
- Array eines statischen Datentyps initialisieren
- Skalar eines dynamischen Datentyps initialisieren
- Array eines dynamischen Datentyps initialisieren
- Größe eines X-Array-Parameters anpassen

Parameter-Informationen holen

Diese Funktion wird vom 3GL-Programm verwendet, um alle erforderlichen Informationen zu Parametern zu erhalten. Diese Informationen werden in einer als `struct parameter_description` bezeichneten, strukturierten Parameterbeschreibung zurückgegeben, siehe oben.

Prototyp:

```
int ncxr_get_parm_info ( int parmnum, void *parmhandle, struct parameter_description *descr );
```

Parameter-Beschreibung:

`parmnum`	Ordnungszahl des Parameters. Diese identifiziert den Parameter der Liste der übergebenen Parameter. Bereich: `0` ... `numparm`–`1`.	
`parmhandle`	Zeiger zur internen Parameter-Struktur	
`descr`	Adresse einer `struct parameter_description`	
`return`	Rückgabewert:	Informationen:
	`0`	OK
	`-1`	Fehlerhafte Parameter-Nummer
	`-2`	Interner Fehler
	`-7`	Schnittstellen-Versionskonflikt

Parameterdaten holen

Diese Funktion wird vom 3GL-Programm verwendet, um die Daten von beliebigen Parametern zu holen.

Natural identifiziert den Parameter über die vorgegebene Parameter-Nummer und schreibt die Parameterdaten unter der gegebenen Pufferadresse in der gegebenen Pufferlänge.

Wenn die Parameterdaten länger als die gegebene Pufferlänge sind, schneidet Natural die Daten bis auf die gegebene Länge ab. Das externe 3GL-Programm kann die Funktion `ncxr_get_parm_info` nutzen, um die Länge der Parameterdaten abzufragen.

Es gibt zwei Funktionen zum Holen von Parameterdaten: `ncxr_get_parm` holt den gesamten Parameter (auch wenn der Parameter ein Array ist), während `ncxr_get_parm_array` das angegebene Array-Element holt.

Wenn vom 3GL-Programm für `buffer` kein Speicher der angegebenen Größe (dynamisch oder statisch) zugewiesen wird, sind die Ergebnisse der Operation nicht vorhersehbar. Natural überprüft dann nur die Pointer auf Gleichheit mit Null.

Wenn Daten bei Variablen des Typs I2/I4/F4/F8 (Pufferlänge ungleich Parameter-Gesamtlänge) abgeschnitten werden, sind die Ergebnisse vom Maschinentyp (Little Endian = höherwertiges Byte vorne /Big Endian = höherwertiges Byte hinten) abhängig. In einigen Anwendungen muss der User Exit programmiert werden, um keine statischen Daten zu verwenden, so dass eine Rekursion möglich wird.

Prototypen:

```
int ncxr_get_parm( int parmnum, void *parmhandle, int buffer_length, void *buffer )

int ncxr_get_parm_array( int parmnum, void *parmhandle, int buffer_length, void *buffer, int *indexes )
```

Diese Funktion ist identisch mit `ncxr_get_parm`, außer dass die Indizes für jede Dimension angegeben werden können. Die Indizes für unbenutzte Dimensionen sollten als Null (0) angegeben werden.

Parameter-Beschreibung:

parmnum	Ordnungszahl des Parameters. Diese identifiziert den Parameter der Liste der übergebenen Parameter. Bereich: 0 ... numparm-1.	
parmhandle	Zeiger zur internen Parameter-Struktur.	
buffer_length	Länge des Puffers, wohin die angeforderten Daten geschrieben werden müssen.	
buffer	Adresse des Puffers, wohin die angeforderten Daten geschrieben werden müssen. Dieser Puffer sollte ausgerichtet werden, um einen leichten Zugriff auf I2/I4/F4/F8-Variablen zu ermöglichen.	
indexes	Array mit Index-Informationen.	
return	Rückgabewert:	Informationen:
	< 0	Fehler beim Einlesen der Informationen.
	-1	Fehlerhafte Parameter-Nummer.
	-2	Interner Fehler.
	-3	Daten wurden abgeschnitten.
	-4	Daten sind kein Array.
	-7	Schnittstellen-Versionskonflikt.
	-100	Index für Dimension 0 liegt außerhalb des zulässigen Bereichs.
	-101	Index für Dimension 1 liegt außerhalb des zulässigen Bereichs.
	-102	Index für Dimension 2 liegt außerhalb des zulässigen Bereichs.
	0	Erfolgreiche Ausführung.
	> 0	Erfolgreiche Ausführung, allerdings sind die Daten nur genau diese Anzahl Bytes lang (Puffer war länger als die Daten).

Operanden-Daten zurückschreiben

Diese Funktionen werden vom 3GL-Programm verwendet, um die Daten auf beliebige Parameter zurückzuschreiben. Natural identifiziert den Parameter über die gegebene Parameter-Nummer und schreibt die Parameterdaten von der gegebenen Pufferadresse in der gegebenen Pufferlänge auf die Parameterdaten.

Wenn die Parameterdaten kürzer als die gegebene Pufferlänge sind, werden die Daten bis auf die Länge der Parameterdaten abgeschnitten, d.h. der Rest des Puffers wird ignoriert. Wenn die Parameterdaten länger als die gegebene Pufferlänge sind, werden die Daten nur in der angegebenen Pufferlänge kopiert, die verbleibenden Parameter bleiben davon unberührt. Dies gilt gleichermaßen für Arrays. Bei dynamischen Variablen als Parameter wird der Parameter auf die angegebene Pufferlänge geändert.

Wenn Daten bei Variablen des Typs I2/I4/F4/F8 (Pufferlänge ungleich Parameter-Gesamtlänge) abgeschnitten werden, sind die Ergebnisse abhängig vom Maschinentyp (Little Endian = höherwertiges Byte vorne, Big Endian = höherwertiges Byte hinten). In einigen Anwendungen muss der User Exit programmiert werden, um keine statischen Daten zu verwenden, so dass eine Rekursion möglich wird.

Prototypen:

```
int ncxr_put_parm        ( int parmnum, void *parmhandle,
                             int buffer_length, void *buffer );
int ncxr_put_parm_array  ( int parmnum, void *parmhandle,
                            int buffer_length, void *buffer,
                           int *indexes );
```

Parameter-Beschreibung:

`parmnum`	Ordnungszahl des Parameters. Diese identifiziert den Parameter der Liste der übergebenen Parameter. Bereich: `0` ... `numparm–1`.	
`parmhandle`	Zeiger zur internen Parameter-Struktur.	
`buffer_length`	Länge der Daten, die in die Adresse des Puffers zurück zu kopieren sind, von wo die Daten herkommen.	
`indexes`	Index information	
`return`	Rückgabewert:	Informationen:
	`< 0`	Fehler beim Zurückkopieren der Informationen.
	`-1`	Fehlerhafte Parameter-Nummer.
	`-2`	Interner Fehler.
	`-3`	Zu viele Daten angegeben. Zurückkopieren erfolgte über die Parameterlänge.
	`-4`	Parameter ist kein Array.
	`-5`	Parameter ist geschützt (konstant oder `AD=O`).
	`-6`	Die Länge der dynamischen Variable konnte aufgrund einer Out of Memory-Bedingung (kein Speicher verfügbar) nicht geändert werden.
	`-7`	Schnittstellen-Versionskonflikt.
	`-13`	Der vorhandene Puffer enthält ein unvollständiges Unicode -Zeichen.
	`-100`	Index für Dimension 0 liegt außerhalb des zulässigen Bereichs.
	`-101`	Index für Dimension 1 liegt außerhalb des zulässigen Bereichs.
	`-102`	Index für Dimension 2 liegt außerhalb des zulässigen Bereichs.
	`0`	Erfolgreiche Ausführung.
	`> 0`	Erfolgreiche Ausführung, allerdings sind die Parameter genau diese Anzahl Bytes lang (Länge des Parameters > gegebene Länge).

Parameter-Set erstellen, initialisieren und löschen

Wenn ein 3GL-Programm ein Natural-Subprogramm aufrufen möchte, muss es einen Parameter-Set erstellen, die den Parametern entspricht, welche das Subprogramm erwartet. Die Funktion `ncxr_create_parm` wird benutzt, um einen Parameter-Set zu erstellen, die mit einem Aufruf an `ncxr_if_callnat` übergeben werden sollen. Der erstellte Parameter-Set wird durch eine transparente Parameter-Struktur dargestellt, wie der Parameter-Set, der an das 3GL-Programm mit dem Statement

CALL INTERFACE4 übergeben wird. Somit kann der neu erstellte Parameter-Set mit den Funktionen ncxr_put_parm* und ncxr_get_parm* wie weiter oben beschrieben bearbeitet werden.

Der neu erstellte Parameter-Set wird noch nicht initialisiert, nachdem die Funktion ncxr_create_parm aufgerufen worden ist. Ein einzelner Parameter wird durch einen Set von unten beschriebenen ncxr_parm_init*-Funktionen für einen spezifischen Datentyp initialisiert. Die Funktionen ncxr_put_parm* und ncxr_get_parm* werden dann benutzt, um auf den Inhalt jedes einzelnen Parameters zuzugreifen. Nachdem der Aufrufende den Parameter-Set abgearbeitet hat, müssen sie die Parameter-Struktur löschen. So sieht dann eine typische Reihenfolge bei der Erstellung und Benutzung eines Sets von Parametern für ein Subprogramm aus, das über ncxr_if4_callnat aufgerufen werden soll:

```
ncxr_create_parm
ncxr_init_ parm*
ncxr_init_ parm*
...
ncxr_put_ parm*
ncxr_put_ parm*
...
ncxr_get_parm_info*
ncxr_get_parm_info*
...
ncxr_if4_callnat
...
ncxr_get_parm_info*
ncxr_get_parm_info*
...
ncxr_get_ parm*
ncxr_get_ parm*
...
ncxr_delete_parm
```

Parameter-Set erstellen

Die Funktion ncxr_create_parm wird benutzt, um einen Parameter-Set zu erstellen, die mit einem Aufruf an ncxr_if_callnat übergeben werden soll.

Prototyp:

```
int ncxr_create_parm( int parmnum, void** pparmhandle )
```

Parameter-Beschreibung:

parmnum	Anzahl der zu erstellenden Parameter.	
pparmhandle	Zeiger zur erstellten Parameter-Struktur.	
return	Rückgabewert:	Information:
	< 0	Fehler
	-1	Fehlerhafte Parameter-Nummer.
	-2	Interner Fehler.
	-6	Out of memory-Bedingung
	0	Erfolgreiche Ausführung.

Parameter-Set löschen

Die Funktion `ncxr_delete_parm` wird benutzt, um einen Parameter-Set zu löschen, der mit `ncxr_create_parm` erstellt wurde.

Prototyp:

```
int ncxr_delete_parm( void* parmhandle )
```

Parameter-Beschreibung:

`parmhandle`	Zeiger zu der zu löschenden Parameter-Struktur.	
`return`	Rückgabewert:	Information:
	`< 0`	Fehler.
	`-2`	Interner Fehler.
	`0`	Erfolgreiche Ausführung.

Skalar eines statischen Datentyps initialisieren

Prototyp:

```
int ncxr_init_parm_s( int parmnum, void *parmhandle,
    char format, int length, int precision, int flags );
```

Parameter-Beschreibung:

`parmnum`	Ordnungszahl des Parameters. Diese identifiziert den Parameter in der Liste der übergebenen Parameter. Bereich: `0` ... `numparm-1`.	
`parmhandle`	Zeiger zur internen Parameter-Struktur.	
`format`	Format des Parameters.	
`length`	Länge des Parameters.	
`precision`	Präzision des Parameters.	
`flags`	Eine Kombination der Flags `IF4_FLG_PROTECTED`	
`return`	Rückgabewert:	Information:
	`< 0`	Fehler.
	`-1`	Fehlerhafte Parameter-Nummer.
	`-2`	Interner Fehler.
	`-6`	Out of memory-Bedingung.
	`-8`	Ungültiges Format.
	`-9`	Ungültige Länge oder Präzision.
	`0`	Erfolgreiche Ausführung.

Array eines statischen Datentyps initialisieren

Prototyp:

```
int ncxr_init_parm_sa( int parmnum, void *parmhandle,
    char format, int length, int precision,
    int dim, int *occ, int flags );
```

Parameter-Beschreibung:

`parmnum`	Ordnungszahl des Parameters. Diese identifiziert den Parameter in der Liste der übergebenen Parameter. Bereich: `0` ... `numparm-1`.	
`parmhandle`	Zeiger zur internen Parameter-Struktur.	
`format`	Format des Parameters.	
`length`	Länge des Parameters.	
`precision`	Präzision des Parameters.	
`dim`	Dimension des Arrays.	
`occ`	Anzahl der Ausprägungen pro Dimension.	

`flags`	Eine Kombination der Flags
	
	`IF4_FLG_PROTECTED`
	`IF4_FLG_LBVAR_0`
	`IF4_FLG_UBVAR_0`
	`IF4_FLG_LBVAR_1`
	`IF4_FLG_UBVAR_1`
	`IF4_FLG_LBVAR_2`
	`IF4_FLG_UBVAR_2`	

`return`	Rückgabewert:	Information:
	`< 0`	Fehler.
	`-1`	Ungültige Parameter-Nummer.
	`-2`	Interner Fehler.
	`-6`	Out of memory-Bedingung.
	`-8`	Ungültiges Format.
	`-9`	Ungültige Länge oder Präzision.
	`-10`	Ungültige Anzahl Dimensionen.
	`-11`	Ungültige Kombination variabler Grenzen.
	`0`	Erfolgreiche Ausführung.

Skalar eines dynamischen Datentyps initialisieren

Prototyp:

```
int ncxr_init_parm_d( int parmnum, void *parmhandle,
    char format, int flags );
```

Parameter-Beschreibung:

`parmnum`	Ordnungszahl des Parameters. Diese identifiziert den Parameter in der Liste der übergebenen Parameter. Bereich: `0` ... `numparm-1`.	
`parmhandle`	Zeiger zur internen Parameter-Struktur.	
`format`	Format des Parameters.	
`flags`	Eine Kombination der Flags `IF4_FLG_PROTECTED`	
`return`	Rückgabewert:	Information:
	`< 0`	Fehler:
	`-1`	Ungültige Parameter-Nummer.
	`-2`	Interner Fehler.
	`-6`	Out of memory-Bedingung.
	`-8`	Ungültiges Format.
	`0`	Erfolgreiche Ausführung.

Array eines dynamischen Datentyps initialisieren

Prototyp:

```
int ncxr_init_parm_da( int parmnum, void *parmhandle,
    char format, int dim, int *occ, int flags );
```

Parameter-Beschreibung:

`parmnum`	Ordnungszahl des Parameters. Diese identifiziert den Parameter in der Liste der übergebenen Parameter. Bereich: `0` ... `numparm-1`.	
`parmhandle`	Zeiger zur internen Parameter-Struktur.	
`format`	Format des Parameters.	
`dim`	Dimension des Arrays.	
`occ`	Anzahl der Ausprägungen pro Dimension.	

`flags`	Eine Kombination der Flags
	
	`IF4_FLG_PROTECTED`
	`IF4_FLG_LBVAR_0`
	`IF4_FLG_UBVAR_0`
	`IF4_FLG_LBVAR_1`
	`IF4_FLG_UBVAR_1`
	`IF4_FLG_LBVAR_2`
	`IF4_FLG_UBVAR_2`	

`return`	Rückgabewert:	Information:
	`< 0`	Fehler.
	`-1`	Ungültige Parameter-Nummer.
	`-2`	Interner Fehler.
	`-6`	Out of memory-Bedingung.
	`-8`	Ungültiges Format.
	`-10`	Ungültige Anzahl Dimensionen.
	`-11`	Ungültige Kombination variabler Grenzen.
	`0`	Erfolgreiche Ausführung.

Größe eines X-Array-Parameters anpassen

Prototype:

```
int ncxr_resize_parm_array( int parmnum, void *parmhandle, int *occ );
```

Parameter-Beschreibung:

`parmnum`	Ordnungszahl des Parameters. Diese identifiziert den Parameter in der Liste der übergebenen Parameter. Bereich: `0` ... `numparm-1`.	
`parmhandle`	Zeiger zur internen Parameter-Struktur.	
`occ`	Neue Anzahl der Ausprägungen pro Dimension	
`return`	Rückgabewert:	Information:
	`< 0`	Fehler.
	`-1`	Ungültige Parameter-Nummer.
	`-2`	Interner Fehler.
	`-6`	Out of memory-Bedingung.
	`-12`	Operand kann größenmäßig nicht angepasst werden (in einer der angegebenen Dimensionen).
	`0`	Erfolgreiche Ausführung.

Alle Funktionsprototypen sind in der Datei *natuser.h* deklariert.