CALL

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

Dieses Kapitel behandelt folgende Themen:

- Funktion
- Syntax-Beschreibung
- Return Code
- User Exits
- 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 oder eine Function aufrufen, wobei im Anschluss daran die Verarbeitung des Natural-Programms mit dem nächsten Statement nach dem `CALL`-Statement fortgesetzt wird.

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

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	**Name des aufgerufenen Programms bzw. der aufgerufenen Function:** Der Name der aufgerufenen Function (*operand1*) kann entweder als Konstante angegeben werden oder — falls je nach Programmlogik verschiedene Programme oder Functions aufgerufen werden sollen — als alphanumerische Variable mit Länge 1 bis 8. Ein Programmname oder Function-Name muss linksbündig in der Variablen stehen.
[USING] ***operand2***	**Parameter:** Das `CALL`-Statement kann bis zu 128 Parameter (*operand2*) enthalten. 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. **Anmerkung:** Wenn eine anwendungsunabhängige Variable (AIV) als Parameter an einen User Exit übergeben wird, gilt die folgende Einschränkung: Falls der User Exit ein Natural-Subprogramm aufruft, das eine neue AIV oder Kontextvariable anlegt, dann ist der Parameter nach der Rückkehr vom Subprogramm ungültig. Dies gilt unabhänig davon, ob die neue AIV bzw. Kontextvariable durch das Subprogramm selbst oder durch ein anderes, direkt oder indirekt von dem Subprogramm aufgerufenes Objekt angelegt wird.

Return Code

Um den Condition Code eines aufgerufenen Programms oder einer Function 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
...
```

User Exits

User Exits werden benötigt, um auf externe Funktionen zugreifen zu können, die mit einem `CALL`-Statement aufgerufen werden. Die User Exits müssen in eine DLL (Dynamic Link Library) gestellt werden. Weitere Informationen zu User Exits finden Sie in der folgenden Datei:

%NATDIR%\%NATVERS%\natural\samples\sysexuex\readme.txt

Voraussetzung: Um Zugang zur die Datei *readme.txt* zu bekommen, müssen Sie bei der Installation von Natural das Merkmal "Samples" ausgewählt haben.

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	32767
Maximale Länge eines Parameters	65535	1 GB
Array-Informationen einlesen	nein	ja
Unterstützung großer und dynamischer Operanden	nein	ja
Parameter-Zugriff über API	nein	ja

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.