CALL

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

This chapter covers the following topics:

- Function
- Syntax Description
- Return Code
- User Exits
- INTERFACE4

For an explanation of the symbols used in the syntax diagram, see *Syntax Symbols*.

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

Belongs to Function Group: *Invoking Programs and Routines*

Function

The `CALL` statement is used to call an external program or function written in another standard programming language from a Natural program and then return to the next statement after the `CALL` statement.

The called program or function may be written in any programming language which supports a standard `CALL` interface. Multiple `CALL` statements to one or more external program or functions may be specified.

Syntax Description

Operand Definition Table:

Operand	Possible Structure					Possible Formats													Referencing Permitted	Dynamic Definition
operand1	C	S				A													yes	no
operand2	C	S	A	G		A	U	N	P	I	F	B	D	T	L	C	G		yes	yes

Syntax Element Description:

Syntax Element	Description
INTERFACE4	**Interface Usage:** The optional keyword INTERFACE4 specifies the type of the interface that is used for the call of the external program. See the section *INTERFACE4* below.
operand1	**Name of Called Function:** The name of the function to be called (*operand1*) can be specified as a constant or - if different functions are to be called dependent on program logic - as an alphanumeric variable of length 1 to 8. A function name must be placed left-justified in the variable.
[USING] *operand2*	**Parameters to be Passed:** The CALL statement may contain up to 128 parameters (*operand2*). One address is passed in the parameter list for each parameter field specified. If a group name is used, the group is converted to individual fields; that is, if a user wishes to specify the beginning address of a group, the first field of the group must be specified. **Note:** If an application-independent variable (AIV) or context variable is passed as a parameter to a user exit, the following restriction applies: if the user exit invokes a Natural subprogram which creates a new AIV or context variable, the parameter is invalid after the return from the subprogram. This is true regardless of whether the new AIV/context variable is created by the subprogram itself or by another object invoked directly or indirectly by the subprogram.

Return Code

The condition code of any called function may be obtained by using the Natural system function RET (Return Code Function).

Example:

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

User Exits

User exits are needed to be able to access external functions that are invoked with a CALL statement. The user exits have to be placed in a DLL (dynamic link library). For further information on the user exits, refer to the following file:

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

The *readme.txt* file is installed together with the samples. As a prerequisite for finding this file, the feature "Samples" must have been selected during the Natural installation.

INTERFACE4

The keyword `INTERFACE4` specifies the type of the interface that is used for the call of the external program. This keyword is optional. If this keyword is specified, the interface, which is defined as `INTERFACE4`, is used for the call of the external program.

The following table lists the differences between the `CALL` statement used with `INTERFACE4` and the one used without `INTERFACE4`:

	CALL statement without keyword INTERFACE4	**CALL statement with keyword INTERFACE4**
Number of parameters possible	128	32767
Maximum data size of one parameter	65535	1 GB
Retrieve array information	no	yes
Support of large and dynamic operands	no	yes
Parameter access via API	no	yes

The following topics are covered below:

- INTERFACE4 - External 3GL Program Interface
- Operand Structure for INTERFACE4
- INTERFACE4 - Parameter Access
- Exported Functions

INTERFACE4 - External 3GL Program Interface

The interface of the external 3GL program is defined as follows, when `INTERFACE4` is specified with the Natural `CALL` statement:

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

`USR_WORD`	`numparm;`	16 bit unsigned short value, containing the total number of transferred operands (*`operand2`*).
`void`	`*parmhandle;`	Pointer to the parameter passing structure.
`void`	`*traditional;`	Check for interface type (if it is not a `NULL` pointer it is the traditional `CALL` interface).

Operand Structure for INTERFACE4

The operand structure of `INTERFACE4` is named `parameter_description` and is defined as follows. The structure is delivered with the header file *natuser.h*.

`struct parameter_description`		
`void *`	`address`	Address of the parameter data, not aligned, `realloc()` and `free()` are not allowed.
`int`	`format`	Field data format: `NCXR_TYPE_ALPHA`, etc. (*natuser.h*).
`int`	`length`	Length (before decimal point, if applicable).
`int`	`precision`	Length after decimal point (if applicable).
`int`	`byte_length`	Length of field in bytes int dimension number of dimensions (`0` to `IF4_MAX_DIM`).
`int`	`dimensions`	Number of dimensions (`0` to `IF4_MAX_DIM`).
`int`	`length_all`	Total data length of array in bytes.

int	flags	Several flag bits combined by bitwise OR operation, meaning:	
		IF4_FLG_PROTECTED:	The parameter is write protected.
		IF4_FLG_DYNAMIC:	The parameter is a dynamic variable.
		IF4_FLG_NOT_CONTIGUOUS:	The array elements are not contiguous (have spaces between them).
		IF4_FLG_AIV:	The parameter is an application-independent variable.
		IF4_FLG_DYNVAR:	The parameter is a dynamic variable.
		IF4_FLG_XARRAY:	The parameter is an X-array.
		IF4_FLG_LBVAR_0:	The lower bound of dimension 0 is variable.
		IF4_FLG_UBVAR_0:	The upper bound of dimension 0 is variable.
		IF4_FLG_LBVAR_1:	The lower bound of dimension 1 is variable.
		IF4_FLG_UBVAR_1:	The upper bound of dimension 1 is variable.
		IF4_FLG_LBVAR_2:	The lower bound of dimension 2 is variable.
		IF4_FLG_UBVAR_2:	The upper bound of dimension 2 is variable.
int	occurrences[IF4_MAX_DIM]	Array occurrences in each dimension.	
int	indexfactors[IF4_MAX_DIM]	Array indexfactors for each dimension.	
void *	dynp	Reserved for internal use.	
void *	pops	Reserved for internal use.	

The address element is null for arrays of dynamic variables and for X-arrays. In these cases, the array data cannot be accessed as a whole, but must be accessed through the parameter access functions described below.

For arrays with fixed bounds of variables with fixed length, the array contents can be accessed directly using the address element. In these cases the address of an array element (i,j,k) is computed as follows (especially if the array elements are not contiguous):

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

If the array has less than 3 dimensions, leave out the last terms.

INTERFACE4 - Parameter Access

A set of functions is available to be used for the access of the parameters. The process flow is as follows:

- The 3GL program is called via the `CALL` statement with the `INTERFACE4` option, and the parameters are passed to the 3GL program as described above.
- The 3GL program can now use the exported functions of Natural, to retrieve either the parameter data itself, or information about the parameter, such as format, length, array information, etc.
- The exported functions can also be used to pass back parameter data.

There are also functions to create and initialize a new parameter set in order to call arbitrary subprograms from a 3GL program. With this technique a parameter access is guaranteed to avoid memory overwrites done by the 3GL program. (Natural's data is safe: memory overwrites within the 3GL program's data are still possible).

Exported Functions

The following topics are covered below:

- Get Parameter Information
- Get Parameter Data
- Write Back Operand Data
- Create, Initialize and Delete a Parameter Set
- Create Parameter Set
- Delete Parameter Set
- Initialize a Scalar of a Static Data Type
- Initialize an Array of a Static Data Type
- Initialize a Scalar of a Dynamic Data Type
- Initialize an Array of a Dynamic Data Type
- Resize an X-array Parameter

Get Parameter Information

This function is used by the 3GL program to receive all necessary information from any parameter. This information is returned in the `struct parameter_description`, which is documented above.

Prototype:

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

Parameter Description:

`parmnum`	Ordinal number of the parameter. This identifies the parameter of the passed parameter list. Range: `0 ... numparm-1`.	
`parmhandle`	Pointer to the internal parameter structure	
`descr`	Address of a `struct parameter_description`	
`return`	Return Value:	Information:
	`0`	OK
	`-1`	Illegal parameter number.
	`-2`	Internal error.
	`-7`	Interface version conflict.

Get Parameter Data

This function is used by the 3GL program to get the data from any parameter.

Natural identifies the parameter by the given parameter number and writes the parameter data to the given buffer address with the given buffer size.

If the parameter data is longer than the given buffer size, Natural will truncate the data to the given length. The external 3GL program can make use of the function `ncxr_get_parm_info`, to request the length of the parameter data.

There are two functions to get parameter data: `ncxr_get_parm` gets the whole parameter (even if the parameter is an array), whereas `ncxr_get_parm_array` gets the specified array element.

If no memory of the indicated size is allocated for "buffer" by the 3GL program (dynamically or statically), results of the operation are unpredictable. Natural will only check for a null pointer.

If data gets truncated for variables of the type I2/I4/F4/F8 (buffer length not equal to the total parameter length), the results depend on the machine type (little endian/big endian). In some applications, the user exit must be programmed to use no static data to make recursion possible.

Prototypes:

```
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 )
```

This function is identical to `ncxr_get_parm`, except that the indexes for each dimension can be specified. The indexes for unused dimensions should be specified as `0`.

Parameter Description:

parmnum	Ordinal number of the parameter. This identifies the parameter of the passed parameter list. Range: 0 ... numparm-1.	
parmhandle	Pointer to the internal parameter structure	
buffer_length	Length of the buffer, where the requested data has to be written to	
buffer	Address of buffer, where the requested data has to be written to. This buffer should be aligned to allow easy access to I2/I4/F4/F8 variables.	
indexes	Array with index information	
return	Return Value:	Information:
	< 0	Error during retrieval of the information:
	-1	Illegal parameter number.
	-2	Internal error.
	-3	Data has been truncated.
	-4	Data is not an array.
	-7	Interface version conflict.
	-100	Index for dimension 0 is out of range.
	-101	Index for dimension 1 is out of range.
	-102	Index for dimension 2 is out of range.
	0	Successful operation.
	> 0	Successful operation, but the data was only this number of bytes long (buffer was longer than the data).

Write Back Operand Data

These functions are used by the 3GL program to write back the data to any parameter. Natural identifies the parameter by the given parameter number and writes the parameter data from the given buffer address with the given buffer size to the parameter data. If the parameter data is shorter than the given buffer size, the data will be truncated to the parameters data length, that is, the rest of the buffer will be ignored. If the parameter data is longer than the given buffer size, the data will be copied only to the given buffer length, the rest of the parameter stays untouched. This applies to arrays in the same way. For dynamic variables as parameters, the parameter is resized to the given buffer length.

If data gets truncated for variables of the type I2/I4/F4/F8 (buffer length not equal to the total parameter length), the results depend on the machine type (little endian/big endian). In some applications, the user exit must be programmed to use no static data to make recursion possible.

Prototypes:

```
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 Description:

parmnum	Ordinal number of the parameter. This identifies the parameter of the passed parameter list. Range: 0 ... numparm-1.	
parmhandle	Pointer to the internal parameter structure.	
buffer_length	Length of the data to be copied back to the address of buffer, where the data comes from.	
indexes	Index information	
return	Return Value:	Information:
	< 0	Error during copying of the information:
	-1	Illegal parameter number.
	-2	Internal error.
	-3	Too much data has been given. The copy back was done with parameter length.
	-4	Parameter is not an array.
	-5	Parameter is protected (constant or AD=O).
	-6	Dynamic variable could not be resized due to an "out of memory" condition.
	-7	Interface version conflict.
	-13	The given buffer includes an incomplete Unicode character.
	-100	Index for dimension 0 is out of range.
	-101	Index for dimension 1 is out of range.
	-102	Index for dimension 2 is out of range.
	0	Successful operation.
	> 0	Successful operation., but the parameter was this number of bytes long (length of parameter greater than given length).

Create, Initialize and Delete a Parameter Set

If a 3GL program wants to call a Natural subprogram, it needs to build a parameter set that corresponds to the parameters the subprogram expects. The function ncxr_create_parm is used to create a set of parameters to be passed with a call to ncxr_if_callnat. The set of parameters created is represented by an opaque parameter handle, like the parameter set that is passed to the 3GL program with the CALL INTERFACE4 statement. Thus, the newly created parameter set can be manipulated with functions ncxr_put_parm* and ncxr_get_parm* as described above.

The newly created parameter set is not yet initialized after having called the function ncxr_create_parm. An individual parameter is initialized to a specific data type by a set of ncxr_parm_init* functions described below. The functions ncxr_put_parm* and ncxr_get_parm* are then used to access the contents of each individual parameter. After the caller has finished with the parameter set, they must delete the parameter handle. Thus, a typical sequence in

creating and using a set of parameters for a subprogram to be called through `ncxr_if4_callnat` will be:

```
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
```

Create Parameter Set

The function `ncxr_create_parm` is used to create a set of parameters to be passed with a call to `ncxr_if_callnat`.

Prototype:

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

Parameter Description:

`parmnum`	Number of parameters to be created.	
`pparmhandle`	Pointer to the created parameter handle.	
`return`	Return Value:	Information:
	< `0`	Error:
	`-1`	Illegal parameter count.
	`-2`	Internal error.
	`-6`	Out of memory condition.
	`0`	Successful operation.

Delete Parameter Set

The function `ncxr_delete_parm` is used to delete a set of parameters that was created with `ncxr_create_parm`.

Prototype:

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

Parameter Description:

`parmhandle`	Pointer to the parameter handle to be deleted.	
`return`	Return Value:	Information:
	< `0`	Error:
	`-2`	Internal error.
	`0`	Successful operation.

Initialize a Scalar of a Static Data Type

Prototype:

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

Parameter Description:

`parmnum`	Ordinal number of the parameter. This identifies the parameter in the passed parameter list. Range: `0 ... numparm-1`.	
`parmhandle`	Pointer to the parameter handle.	
`format`	Format of the parameter.	
`length`	Length of the parameter.	
`precision`	Precision of the parameter.	
`flags`	`IF4_FLG_PROTECTED`	
`return`	Return Value:	Information:
	< `0`	Error:
	`-1`	Invalid parameter number.
	`-2`	Internal error.
	`-6`	Out of memory condition.
	`-8`	Invalid format.
	`-9`	Invalid length or precision.
	`0`	Successful operation.

Initialize an Array of a Static Data Type

Prototype:

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

Parameter Description:

`parmnum`	Ordinal number of the parameter. This identifies the parameter in the passed parameter list. Range: `0 ... numparm-1`.	
`parmhandle`	Pointer to the parameter handle.	
`format`	Format of the parameter.	
`length`	Length of the parameter.	
`precision`	Precision of the parameter.	
`dim`	Dimension of the array.	
`occ`	Number of occurrences per dimension.	

`flags`	A combination of the 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`	Return Value:	Information:
	< `0`	Error:
	`-1`	Invalid parameter number.
	`-2`	Internal error.
	`-6`	Out of memory condition.
	`-8`	Invalid format.
	`-9`	Invalid length or precision.
	`-10`	Invalid dimension count.
	`-11`	Invalid combination of variable bounds.
	`0`	Successful operation.

Initialize a Scalar of a Dynamic Data Type

Prototype:

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

Parameter Description:

`parmnum`	Ordinal number of the parameter. This identifies the parameter in the passed parameter list. Range: `0 ... numparm-1`.	
`parmhandle`	Pointer to the parameter handle.	
`format`	Format of the parameter.	
`flags`	`IF4_FLG_PROTECTED`	
`return`	Return Value:	Information:
	< `0`	Error:
	`-1`	Invalid parameter number.
	`-2`	Internal error.
	`-6`	Out of memory condition.
	`-8`	Invalid format.
	`0`	Successful operation.

Initialize an Array of a Dynamic Data Type

Prototype:

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

Parameter Description:

parmnum	Ordinal number of the parameter. This identifies the parameter in the passed parameter list. Range: 0 ... numparm-1.	
parmhandle	Pointer to the parameter handle.	
format	Format of the parameter.	
dim	Dimension of the array.	
occ	Number of occurrences per dimension.	
flags	A combination of the 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	Return Value:	Information:
	< 0	Error:
	-1	Invalid parameter number.
	-2	Internal error.
	-6	Out of memory condition.
	-8	Invalid format.
	-10	Invalid dimension count.
	-11	Invalid combination of variable bounds.
	0	Successful operation.

Resize an X-array Parameter

Prototype:

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

Parameter Description:

parmnum	Ordinal number of the parameter. This identifies the parameter in the passed parameter list. Range: 0 ... numparm-1.	
parmhandle	Pointer to the parameter handle.	
occ	New number of occurrences per dimension.	
return	Return Value:	Information:
	< 0	Error:
	-1	Invalid parameter number.
	-2	Internal error.
	-6	Out of memory condition.
	-12	Operand is not resizable (in one of the specified dimensions).
	0	Successful operation.

All function prototypes are declared in the file natuser.h.