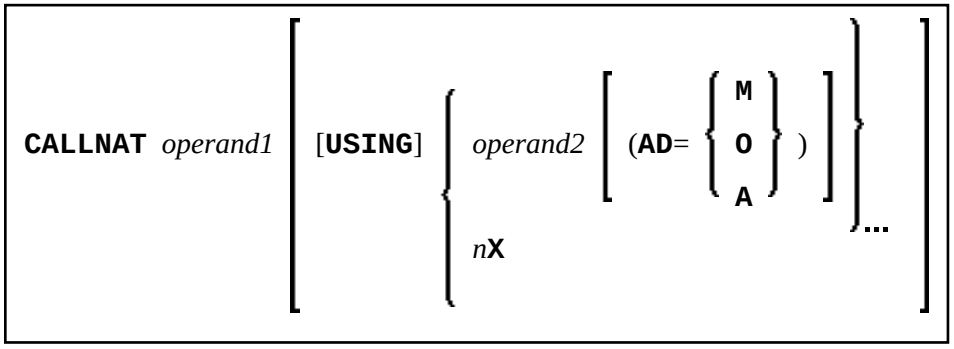
CALLNAT



This chapter covers the following topics:

- Function
- Syntax Description
- Parameter Transfer with Dynamic Variables
- Examples

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

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

Belongs to Function Group: *Invoking Programs and Routines*

Function

The CALLNAT statement is used to invoke a Natural subprogram for execution. (A Natural subprogram can only be invoked via a CALLNAT statement; it cannot be executed by itself.)

When the CALLNAT statement is executed, the execution of the invoking object (that is, the object containing the CALLNAT statement) will be suspended and the invoked subprogram will be executed. The execution of the subprogram continues until either its END statement is reached or processing of the subprogram is stopped by an ESCAPE ROUTINE statement being executed. In either case, processing of the invoking object will then continue with the statement following the CALLNAT statement.

Notes:

1. A subprogram can in turn invoke other subprograms.
2. A subprogram has no access to the global data area used by the invoking object. If a subprogram in turn invokes a subroutine or helproutine, it can establish its own global data area to be shared with the subroutine/helproutine.

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	O	yes	yes

Syntax Element Description:

Syntax Element	Description
operand1	**Subprogram to be Invoked:** As *operand1*, you specify the name of the subprogram to be invoked. The name may be specified either as a constant of 1 to 8 characters, or - if different subprograms are to be called dependent on program logic - as an alphanumeric variable of length 1 to 8. The subprogram name may contain an ampersand (&); at execution time, this character will be replaced by the one-character code corresponding to the current value of the system variable *LANGUAGE. This makes it possible, for example, to invoke different subprograms for the processing of input, depending on the language in which input is provided.

Syntax Element	Description
operand2	**Parameters:** If parameters are passed to the subprogram, the structure of the parameter list must be defined in a DEFINE DATA PARAMETER statement. The parameters specified with the CALLNAT statement are the only data available to the subprogram from the invoking object. By default, the parameters are passed *by reference*, that is, the data are transferred via address parameters, the parameter values themselves are not moved. However, it is also possible to pass parameters *by value*, that is, pass the actual parameter values. To do so, you define these fields in the DEFINE DATA PARAMETER statement of the subprogram with the option BY VALUE or BY VALUE RESULT (see *parameter-data-definition* in the description of the DEFINE DATA statement). ● If parameters are passed *by reference*, the following applies: The sequence, format and length of the parameters in the invoking object must match exactly the sequence, format and length of the DEFINE DATA PARAMETER structure in the invoked subprogram. The names of the variables in the invoking object and the invoked subprogram may be different. ● If parameters are passed *by value*, the following applies: The sequence of the parameters in the invoking object must match exactly the sequence in the DEFINE DATA PARAMETER structure of the invoked subprogram. Formats and lengths of the variables in the invoking object and the subprogram may be different; however, they have to be data transfer compatible; see the corresponding table in the section *Rules for Arithmetic Assignments*, *Data Transfer* in the *Programming Guide*. The names of the variables in the invoking object and the subprogram may be different. If parameter values that have been modified in the subprogram are to be passed back to the invoking object, you have to define these fields with BY VALUE RESULT. When BY VALUE is specified without RESULT, it is not possible to pass modified parameter values back to the invoking object (regardless of the AD specification; see also below). **Note:** With BY VALUE, an internal copy of the parameter values is created. The subprogram accesses this copy and can modify it, but this will not affect the original parameter values in the invoking object. With BY VALUE RESULT, an internal copy is likewise created, however, after termination of the subprogram, the original parameter values are overwritten by the (modified) values of the copy. For both ways of passing parameters, the following applies: If a group is specified as *operand2*, the individual fields contained in that group are passed to the subprogram; that is, for each of these fields a corresponding field must be defined in the subprogram's parameter data area. In the parameter data area of the invoked subprogram, a redefinition of groups is only permitted within a REDEFINE block. If an array is passed, its number of dimensions and occurrences in the subprogram's parameter data area must be the same as in the CALLNAT parameter list. **Note:** If multiple occurrences of an array that is defined as part of an indexed group are passed with the CALLNAT statement, the corresponding fields in the subprogram's parameter data area must not be redefined, as this would lead to the wrong addresses being passed. When the option PCHECK of the COMPOPT command is set to ON, the compiler will check the number, format, length and array index bounds of the parameters that are specified in a CALLNAT statement. Also, the OPTIONAL feature of the DEFINE DATA PARAMETER statement is considered in the parameter check.

Syntax Element	Description	
AD=	**Attribute Definition:** If *operand2* is a variable, you can mark it in one of the following ways:	
	AD=O	Non-modifiable, see session parameter AD=O. **Note:** Internally, AD=O is processed in the same way as BY VALUE (see *parameter-data-definition* in the description of the DEFINE DATA statement).
	AD=M	Modifiable, see session parameter AD=M. This is the default setting.
	AD=A	Input only, see session parameter AD=A.
	If *operand2* is a constant, AD cannot be explicitly specified. For constants AD=O always applies.	
*n*X	**Parameters to be Skipped:** With the notation *n*X you can specify that the next *n* parameters are to be skipped (for example, 1X to skip the next parameter, or 3X to skip the next three parameters); this means that for the next *n* parameters no values are passed to the subprogram. The possible range of values for *n* is 1 - 4096. A parameter that is to be skipped must be defined with the keyword OPTIONAL in the subprogram's DEFINE DATA PARAMETER statement. OPTIONAL means that a value can - but need not - be passed from the invoking object to such a parameter.	

Parameter Transfer with Dynamic Variables

Dynamic variables may be passed as parameters to a called program object (CALLNAT, PERFORM). A call by reference is possible because the value space of a dynamic variable is contiguous. A call by value causes an assignment with the variable definition of the caller as the source operand and the parameter definition as the destination operand. In addition, a call by value result causes the movement to change to the opposite direction. When using a call-by-reference, both definitions must be DYNAMIC. If only one of them is DYNAMIC, a runtime error is raised. In case of a call by value (result) all combinations are possible.

The following table illustrates the valid combinations of statically and dynamically defined variables of the caller, and statically and dynamically defined parameters concerning the parameter transfer.

Call By Reference

operand2 of caller	Parameter definition	
	Static	Dynamic
Static	yes	no
Dynamic	no	yes

The formats of the dynamic variables A or B must match.

Call by Value (Result)

operand2 of caller	Parameter definition	
	Static	Dynamic
Static	yes	yes
Dynamic	yes	yes

Note:
When using static/dynamic or dynamic/static definitions, a value truncation may occur according to the data transfer rules of the appropriate assignments.

Examples

- Example 1
- Example 2

Example 1

Calling Program:

```
** Example 'CNTEX1': CALLNAT
************************************************************************
DEFINE DATA LOCAL
1 #FIELD1 (N6)
1 #FIELD2 (A20)
1 #FIELD3 (A10)
END-DEFINE
*
CALLNAT 'CNTEX1N' #FIELD1 (AD=M) #FIELD2 (AD=O) #FIELD3 'P4 TEXT'
*
WRITE '=' #FIELD1 '=' #FIELD2 '=' #FIELD3
*
END
```

Called Subprogram CNTEX1N:

```
** Example 'CNTEX1N': CALLNAT  (called by CNTEX1)
************************************************************************
DEFINE DATA PARAMETER
1 #FIELDA (N6)
1 #FIELDB (A20)
1 #FIELDC (A10)
1 #FIELDD (A7)
END-DEFINE
*
*
#FIELDA := 4711
*
#FIELDB := 'HALLO'
*
#FIELDC := 'ABC'
```

```
*
WRITE '=' #FIELDA '=' #FIELDB '=' #FIELDC '=' #FIELDD
*
END
```

Example 2

Calling Program:

```
** Example 'CNTEX2': CALLNAT
************************************************************************
DEFINE DATA LOCAL
1 #ARRAY1  (N4/1:10,1:10)
1 #NUM     (N2)
END-DEFINE
*
*
CALLNAT 'CNTEX2N' #ARRAY1 (2:5,*)
*
FOR #NUM 1 TO 10
  WRITE #NUM #ARRAY1(#NUM,1:10)
END-FOR
*
END
```

Called Subprogram CNTEX2N:

```
** Example 'CNTEX2N': CALLNAT  (called by CNTEX2)
************************************************************************
DEFINE DATA
PARAMETER
1 #ARRAY (N4/1:4,1:10)
LOCAL
1 I      (I2)
END-DEFINE
*
*
FOR I 1 10
  #ARRAY(1,I) := I
  #ARRAY(2,I) := 100 + I
  #ARRAY(3,I) := 200 + I
  #ARRAY(4,I) := 300 + I
END-FOR
*
END
```