

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 as described under Parameter-Data-Definition in the section DEFINE DATA.

- 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 Natural Reference documentation). 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. With BY VALUE (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.

AD=

If *operand2* is a variable, you can mark it in one of the following ways:

AD=O	non-modifiable
AD=M	modifiable
AD=A	input only

The default setting for AD= is AD=M.

If *operand2* is a constant, AD cannot be explicitly specified. For constants AD=O always applies.

AD=M

By default, the passed value of a parameter can be changed in the subprogram and the changed value passed back to the invoking object, where it overwrites the original value.

Exception: For a field defined with BY VALUE in the subprogram's parameter data area, no value is passed back.

AD=O

If you mark a parameter with AD=O, the passed value can be changed in the subprogram, but the changed value cannot be passed back to the invoking object; that is, the field in the invoking object retains its original value.

Note:

Internally, AD=O is processed in the same way as BY VALUE (see the section parameter-data-definition in the description of the DEFINE DATA statement).

AD=A

If you mark a parameter with AD=A, its value will not be passed to the subprogram, but it will receive a value from the subprogram. This may be useful for remote subprograms executed via Natural RPC in a client/server environment to reduce the load of data sent. If a subprogram is executed locally, AD=A fields will be reset to empty before the subprogram is invoked.

For a field defined with BY VALUE in the subprogram's parameter data area, the invoking object cannot receive a value. In this case, AD=A only causes the field to be reset to empty before the subprogram is invoked.

nX

Note:

This notation is not available on mainframe computers.

With the notation nX 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.

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.

Other Considerations

A subprogram can in turn invoke other subprograms.

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.

Parameter Transfer with Dynamic Variables

Dynamic variables may be passed as parameters to a called program object (CALLNAT, PERFORM).

Call-by-reference is possible because the value space of a dynamic variable is contiguous. 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. Call-by-value result causes additionally the movement in the opposite direction. In case of call-by-reference both definitions must be DYNAMIC. If only one of them is DYNAMIC, a runtime error is raised.

In case of 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:

In case of static/dynamic or dynamic/static definitions, a value truncation may occur according to the data transfer rules of the appropriate assignments.

Example 1

Invoking Program:

```

/* EXAMPLE 'CNTEX1': CALLNAT
/*****
/* MAIN PROGRAM 'MAINP1'
/*****
DEFINE DATA LOCAL
1 #FIELD1 (N6)
1 #FIELD2 (A20)
1 #FIELD3 (A10)
END-DEFINE
/*****
CALLNAT 'SUBP1' #FIELD1 (AD=M) #FIELD2 (AD=O) #FIELD3 'P4 TEXT'
/* ...
END
    
```

Invoked Subprogram:

```
/* SUBPROGRAM 'SUBP1'
/*****
DEFINE DATA PARAMETER
1 #FIELDA (N6)
1 #FIELDB (A20)
1 #FIELD C (A10)
1 #FIELD D (A7)
END-DEFINE
/*****
/* ...
END
```

Example 2

Invoking Program:

```
/* EXAMPLE 'CNTEX2': CALLNAT
/*****
/* MAIN PROGRAM 'MAINP2'
/*****
DEFINE DATA LOCAL
1 #ARRAY1 (A3/1:10,1:10)
END-DEFINE
CALLNAT 'SUBP2' #ARRAY1 (2:5,*)
/* ...
```

Invoked Subprogram:

```
/* SUBPROGRAM 'SUBP2'
/*****
DEFINE DATA PARAMETER
1 #ARRAY (A3/1:4,1:10)
END-DEFINE
/*****
/* ...
END
```