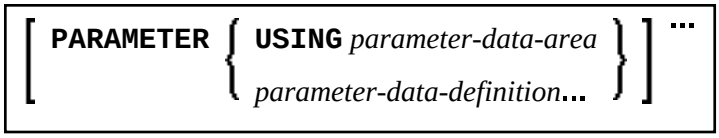
Defining Parameter Data

General syntax of DEFINE DATA PARAMETER:



This chapter covers the following topics:

- Function
- Restrictions
- Syntax Description

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

Function

The DEFINE DATA PARAMETER statement is used to define the data elements that are to be used as incoming parameters in a Natural subprogram, external subroutine or helproutine. These parameters can be defined within the statement itself (see *Parameter Data Definition* below); or they can be defined outside the program in a parameter data area (PDA), with the statement referencing that data area.

Restrictions

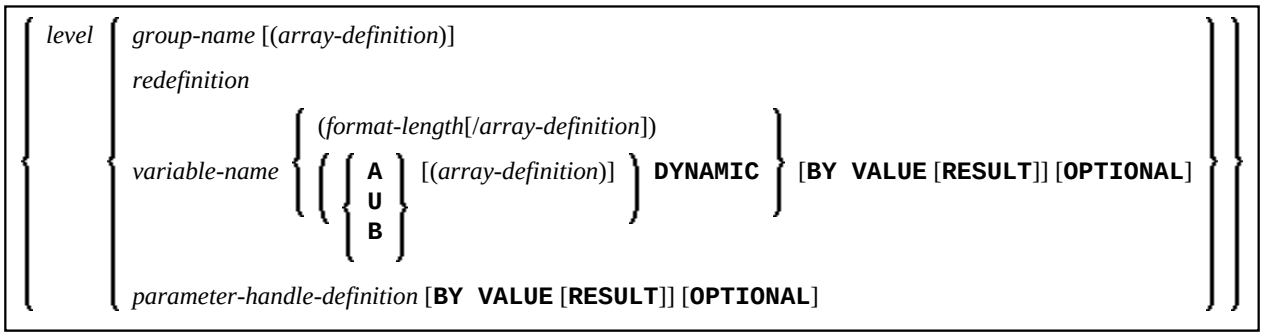
- Parameter data elements must not be assigned initial or constant values, and they must not have edit mask (EM), header (HD) or print mode (PM) definitions (see also *EM, HD, PM Parameters for Field/Variable*).
- The parameter data area and the objects which reference it must be contained in the same library (or in a steplib).

Syntax Description

USING ***parameter-data-area***	The name of the *parameter-data-area* that contains data elements which are used as parameters in a subprogram, external subroutine or dialog.
parameter-data-definition	Instead of defining a parameter data area, parameter data can also be defined directly within a program or routine; see *Parameter Data Definition* below.
END-DEFINE	The Natural reserved word END-DEFINE must be used to end the DEFINE DATA statement.

Parameter Data Definition

For direct parameter data definition, the following syntax applies:



Syntax Element Description:

level	Level number is a 1- or 2-digit number in the range from 01 to 99 (the leading zero is optional) used in conjunction with field grouping. Fields assigned a level number of 02 or greater are considered to be a part of the immediately preceding group which has been assigned a lower level number. The definition of a group enables reference to a series of fields (may also be only 1 field) by using the group name. With certain statements (`CALL`, `CALLNAT`, `RESET`, `WRITE`, etc.), you may specify the group name as a shortcut to reference the fields contained in the group. A group may consist of other groups. When assigning the level numbers for a group, no level numbers may be skipped.
group-name	The name of a group. The name must adhere to the rules for defining a Natural variable name. See also the following sections: • *Naming Conventions for User-Defined Variables* in the *Using Natural* documentation. • *Qualifying Data Structures* in the *Programming Guide*.
array-definition	With an *array-definition*, you define the lower and upper bounds of dimensions in an array-definition. See *Array Dimension Definition* and *Variable Arrays in a Parameter Data Area*.
redefinition	A *redefinition* may be used to redefine a group or a single field/variable (that is a scalar or an array). See *Redefinition*. **Note:** In a *parameter-data-definition*, a "redefinition" of groups is only permitted within a `REDEFINE` block.
variable-name	The name to be assigned to the variable. Rules for Natural variable names apply. For information on naming conventions for user-defined variables, see *Naming Conventions for User-Defined Variables* in the *Using Natural* documentation.

format-length	The format and length of the field. For information on format/length definition of user-defined variables, see *Format and Length of User-Defined Variables* in the *Programming Guide*.
A, U or B	Data type: alphanumeric (A), Unicode (U) or binary (B) for dynamic variable.
DYNAMIC	A parameter may be defined as `DYNAMIC`. For more information on processing dynamic variables, see *Introduction to Dynamic Variables and Fields*.
	Call Mode: Depending on whether call-by-reference, call-by-value or call-by-value-result is used, the appropriate transfer mechanism is applicable. For further information, see the `CALLNAT` statement.
(without BY VALUE)	**Call-by-reference:** Call-by-reference is active by default when you omit the `BY VALUE` keywords. In this case, a parameter is passed to a subprogram/subroutine by reference (that is, via its address); therefore a field specified as parameter in a `CALLNAT`/`PERFORM` statement must have the same format/length as the corresponding field in the invoked subprogram/subroutine.
BY VALUE	**Call-by-value:** When you specify `BY VALUE`, a parameter is passed to a subprogram/subroutine by value; that is, the actual parameter value (instead of its address) is passed. Consequently, the field in the subprogram/subroutine need not have the same format/length as the `CALLNAT`/`PERFORM` parameter. The formats/lengths must only be data transfer compatible. For data transfer compatibility, the *Rules for Arithmetic Assignment/Data Transfer* apply (see *Programming Guide*). `BY VALUE` allows you, for example, to increase the length of a field in a subprogram/subroutine (if this should become necessary due to an enhancement of the subprogram/subroutine) without having to adjust any of the objects that invoke the subprogram/subroutine.
	Example of BY VALUE:

```
* Program
DEFINE DATA LOCAL
1 #FIELDA (P5)
...
END-DEFINE
...
CALLNAT 'SUBR01' #FIELDA
...
```

```
* Subroutine SUBR01
DEFINE DATA PARAMETER
1 #FIELDB (P9) BY VALUE
END-DEFINE
...
```

BY VALUE RESULT	**Call-by-value-result:** While `BY VALUE` applies to a parameter passed to a subprogram/subroutine, `BY VALUE RESULT` causes the parameter to be passed by value in both directions; that is, the actual parameter value is passed from the invoking object to the subprogram/subroutine and, on return to the invoking object, the actual parameter value is passed from the subprogram/subroutine back to the invoking object. With `BY VALUE RESULT`, the formats/lengths of the fields concerned must be data transfer compatible in both directions.
OPTIONAL	For a parameter defined without `OPTIONAL` (default), a value *must* be passed from the invoking object. For a parameter defined with `OPTIONAL`, a value can, but need not be passed from the invoking object to this parameter. In the invoking object, the notation *n*`X` is used to indicate parameters which are skipped, that is, for which no values are passed. With the `SPECIFIED` option you can find out at run time whether an optional parameter has been defined or not.
parameter-handle-definition	See the section *Parameter Handle Definition* below.

Parameter Handle Definition

Syntax of *parameter-handle-definition*:

handle-name [(*array-definition*)] **HANDLE OF OBJECT**

Syntax Element Description:

handle-name	The name to be assigned to the handle; the naming conventions for user-defined variables apply; see *Naming Conventions for User-Defined Variables* in the *Using Natural* documentation..
HANDLE OF OBJECT	Is used in conjunction with NaturalX as described in the section *NaturalX* of the *Programming Guide*.
array-definition	With an *array-definition*, you define the lower and upper bounds of dimensions in an array-definition. See *Array Dimension Definition*.