

# Defining Parameter Data

General syntax of `DEFINE DATA PARAMETER`:

$\left[ \text{PARAMETER} \left\{ \begin{array}{l} \text{USING } \textit{parameter-data-area} \\ \textit{parameter-data-definition...} \end{array} \right\} \right] \dots$
---

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 help routine. 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

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

## Parameter Data Definition

For direct parameter data definition, the following syntax applies:

$  \left\{ \begin{array}{l}  \textit{level} \left\{ \begin{array}{l}  \textit{group-name} [(\textit{array-definition})] \\  \textit{redefinition} \\  \textit{variable-name} \left\{ \begin{array}{l}  (\textit{format-length}[/\textit{array-definition}]) \\  \left( \left( \left\{ \begin{array}{l}  \text{A} \\  \text{U} \\  \text{B}  \end{array} \right\} [(\textit{array-definition})] \right) \text{DYNAMIC} \left\{ \begin{array}{l}  [\text{BY VALUE} [\text{RESULT}]] [\text{OPTIONAL}]  \end{array} \right\}  \end{array} \right\} \\  \textit{parameter-handle-definition} [\text{BY VALUE} [\text{RESULT}]] [\text{OPTIONAL}]  \end{array} \right.  \end{array}  \right.  $
--

Syntax Element Description:

<b><i>level</i></b>	<p>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.</p> <p>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.</p> <p>A group may consist of other groups. When assigning the level numbers for a group, no level numbers may be skipped.</p>
<b><i>group-name</i></b>	<p>The name of a group. The name must adhere to the rules for defining a Natural variable name. See also the following sections:</p> <ul style="list-style-type: none"> <li>● <i>Naming Conventions for User-Defined Variables</i> in the <i>Using Natural</i> documentation.</li> <li>● <i>Qualifying Data Structures</i> in the <i>Programming Guide</i>.</li> </ul>
<b><i>array-definition</i></b>	<p>With an <i>array-definition</i>, you define the lower and upper bounds of dimensions in an array-definition. See <i>Array Dimension Definition</i> and <i>Variable Arrays in a Parameter Data Area</i>.</p>
<b><i>redefinition</i></b>	<p>A <i>redefinition</i> may be used to redefine a group or a single field/variable (that is a scalar or an array). See <i>Redefinition</i>.</p> <p><b>Note:</b> In a <i>parameter-data-definition</i>, a "redefinition" of groups is only permitted within a REDEFINE block.</p>
<b><i>variable-name</i></b>	<p>The name to be assigned to the variable. Rules for Natural variable names apply. For information on naming conventions for user-defined variables, see <i>Naming Conventions for User-Defined Variables</i> in the <i>Using Natural</i> documentation.</p>

<i>format-length</i>	The format and length of the field. For information on format/length definition of user-defined variables, see <i>Format and Length of User-Defined Variables</i> in the <i>Programming Guide</i> .			
<b>A, U or B</b>	Data type: alphanumeric (A), Unicode (U) or binary (B) for dynamic variable.			
<b>DYNAMIC</b>	A parameter may be defined as DYNAMIC. For more information on processing dynamic variables, see <i>Introduction to Dynamic Variables and Fields</i> .			
	<b>Call Mode:</b>  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.			
<b>(without BY VALUE)</b>	<b>Call-by-reference:</b>  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.			
<b>BY VALUE</b>	<p><b>Call-by-value:</b></p> <p>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 <i>Rules for Arithmetic Assignment/Data Transfer</i> apply (see <i>Programming Guide</i>).</p> <p>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.</p> <p><b>Example of BY VALUE:</b></p> <table border="1" data-bbox="537 1507 1393 1749"> <tr> <td> <pre>* Program DEFINE DATA LOCAL 1 #FIELD A (P5) ... END-DEFINE ... CALLNAT 'SUBR01' #FIELD A ...</pre> </td> <td> <pre>* Subroutine SUBR01 DEFINE DATA PARAMETER 1 #FIELD B (P9) BY VALUE END-DEFINE ...</pre> </td> </tr> </table>		<pre>* Program DEFINE DATA LOCAL 1 #FIELD A (P5) ... END-DEFINE ... CALLNAT 'SUBR01' #FIELD A ...</pre>	<pre>* Subroutine SUBR01 DEFINE DATA PARAMETER 1 #FIELD B (P9) BY VALUE END-DEFINE ...</pre>
<pre>* Program DEFINE DATA LOCAL 1 #FIELD A (P5) ... END-DEFINE ... CALLNAT 'SUBR01' #FIELD A ...</pre>	<pre>* Subroutine SUBR01 DEFINE DATA PARAMETER 1 #FIELD B (P9) BY VALUE END-DEFINE ...</pre>			

<b>BY VALUE RESULT</b>	<p><b>Call-by-value-result:</b></p> <p>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.</p> <p>With BY VALUE RESULT, the formats/lengths of the fields concerned must be data transfer compatible in both directions.</p>
<b>OPTIONAL</b>	<p>For a parameter defined without OPTIONAL (default), a value <i>must</i> be passed from the invoking object.</p> <p>For a parameter defined with OPTIONAL, a value can, but need not be passed from the invoking object to this parameter.</p> <p>In the invoking object, the notation <i>nX</i> is used to indicate parameters which are skipped, that is, for which no values are passed.</p> <p>With the SPECIFIED option you can find out at run time whether an optional parameter has been defined or not.</p>
<i>parameter-handle-definition</i>	See the section <i>Parameter Handle Definition</i> below.

## Parameter Handle Definition

Syntax of *parameter-handle-definition*:

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

Syntax Element Description:

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