

Defining Parameter Data

General syntax of `DEFINE DATA PARAMETER`:

```
[ PARAMETER { USING parameter-data-area } ] ...  
                { parameter-data-definition... }
```

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, help routine or function. 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

Syntax Element	Description
USING <i>parameter-data-area</i>	PDA Name: 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>	Direct Parameter Data Definition: Instead of defining a parameter data area, parameter data can also be defined directly within a program or routine. See <i>Direct Parameter Data Definition</i> below.
END-DEFINE	End of DEFINE DATA Statement: The Natural reserved word END-DEFINE must be used to end the DEFINE DATA statement.

Direct Parameter Data Definition

For direct parameter data definition, the following syntax applies:

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

Syntax Element Description:

Syntax Element	Description
<i>level</i>	<p>Level Number:</p> <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>
<i>group-name</i>	<p>Group Name:</p> <p>The name of a group. The name must adhere to the rules for defining a Natural variable name.</p> <p>See also the following sections:</p> <ul style="list-style-type: none"> ● <i>Naming Conventions for User-Defined Variables in Using Natural.</i> ● <i>Qualifying Data Structures in the Programming Guide.</i>
<i>array-definition</i>	<p>Array Dimension Definition:</p> <p>With an <i>array-definition</i>, you define the lower and upper bounds of dimensions in an array-definition.</p> <p>For further information, see <i>Array Dimension Definition</i> and <i>Variable Arrays in a Parameter Data Area</i>.</p>
<i>redefinition</i>	<p>Redefinition:</p> <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>Note: In a <i>parameter-data-definition</i>, a redefinition of groups is only permitted within a REDEFINE block.</p>

Syntax Element	Description
<i>variable-name</i>	<p>Variable Name:</p> <p>The name to be assigned to the variable. Rules for Natural variable names apply. For information on naming conventions for user-defined variables.</p> <p>For further information, see <i>Naming Conventions for User-Defined Variables</i> in <i>Using Natural</i>.</p>
<i>format-length</i>	<p>Format/Length Definition:</p> <p>The format and length of the field. For information on format/length definition of user-defined variables.</p> <p>For further information, see <i>Format and Length of User-Defined Variables</i> in the <i>Programming Guide</i>.</p>
A, U or B	<p>Data Type:</p> <p>Alphanumeric (A), Unicode (U) or binary (B) for dynamic variable.</p>
DYNAMIC	<p>DYNAMIC Option:</p> <p>A parameter may be defined as DYNAMIC. For further information on processing dynamic variables, see <i>Introduction to Dynamic Variables and Fields</i> in the <i>Programming Guide</i>.</p>
	<p>Call Mode:</p> <p>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.</p>
(without BY VALUE)	<p>Call-by-Reference:</p> <p>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.</p>

Syntax Element	Description		
BY VALUE	<p>Call-by-Value:</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</i> and <i>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>Example of BY VALUE:</p> <table border="0" data-bbox="667 856 1393 1100"> <tr> <td data-bbox="667 863 1036 1100"> <pre>* Program DEFINE DATA LOCAL 1 #FIELDA (P5) ... END-DEFINE ... CALLNAT 'SUBR01' #FIELDA ...</pre> </td> <td data-bbox="1036 863 1393 1100"> <pre>* Subroutine SUBR01 DEFINE DATA PARAMETER 1 #FIELDDB (P9) BY VALUE END-DEFINE ...</pre> </td> </tr> </table>	<pre>* Program DEFINE DATA LOCAL 1 #FIELDA (P5) ... END-DEFINE ... CALLNAT 'SUBR01' #FIELDA ...</pre>	<pre>* Subroutine SUBR01 DEFINE DATA PARAMETER 1 #FIELDDB (P9) BY VALUE END-DEFINE ...</pre>
<pre>* Program DEFINE DATA LOCAL 1 #FIELDA (P5) ... END-DEFINE ... CALLNAT 'SUBR01' #FIELDA ...</pre>	<pre>* Subroutine SUBR01 DEFINE DATA PARAMETER 1 #FIELDDB (P9) BY VALUE END-DEFINE ...</pre>		
BY VALUE RESULT	<p>Call-by-Value-Result:</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>		

Syntax Element	Description
OPTIONAL	<p>Optional Parameters:</p> <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>	<p>Parameter Handle Definition:</p> <p>See the section <i>Parameter Handle Definition</i> below.</p>

Parameter Handle Definition

Syntax of *parameter-handle-definition*:

<i>handle-name</i> [(<i>array-definition</i>)] HANDLE OF OBJECT
--

Syntax Element Description:

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