Use and Structure of DEFINE DATA Statement

The first statement in a Natural program written in structured mode must always be a `DEFINE DATA` statement which is used to define fields for use in a program.

This chapter covers the following topics:

- Field Definitions in DEFINE DATA Statement
- Defining Fields within a DEFINE DATA Statement
- Defining Fields in a Separate Data Area
- Structuring a DEFINE DATA Statement Using Level Numbers
- Storage Alignment

For information on structural indentation of a source program, see the Natural system command `STRUCT`.

Field Definitions in DEFINE DATA Statement

In the `DEFINE DATA` statement, you define all the fields - database fields as well as user-defined variables - that are to be used in the program.

There are two ways to define the fields:

- The fields can be defined within the `DEFINE DATA` statement itself (see below).
- The fields can be defined outside the program in a local or global data area, with the `DEFINE DATA` statement referencing that data area (see below).

If fields are used by multiple programs/routines, they should be defined in a data area outside the programs.

For a clear application structure, it is usually better to define fields in data areas outside the programs.

Data areas are created and maintained with the data area editor, which is described in the *Editors* documentation.

In the first example below, the fields are defined within the `DEFINE DATA` statement of the program. In the second example, the same fields are defined in a local data area (LDA), and the `DEFINE DATA` statement only contains a reference to that data area.

Defining Fields within a DEFINE DATA Statement

The following example illustrates how fields can be defined within the `DEFINE DATA` statement itself:

```
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 PERSONNEL-ID
1 #VARI-A (A20)
1 #VARI-B (N3.2)
1 #VARI-C (I4)
END-DEFINE
...
```

Defining Fields in a Separate Data Area

The following example illustrates how fields can be defined in a local data area (LDA):

Program:

```
DEFINE DATA LOCAL
  USING LDA39
END-DEFINE
...
```

Local Data Area LDA39:

```
I T L Name                            F Leng Index/Init/EM/Name/Comment
- - - --------------------------------- - ---- ---------------------------------
  V 1 VIEWEMP                                 EMPLOYEES
    2 NAME                            A   20
    2 FIRST-NAME                      A   20
    2 PERSONNEL-ID                    A    8
    1 #VARI-A                         A   20
    1 #VARI-B                         N  3.2
    1 #VARI-C                         I    4
```

Structuring a DEFINE DATA Statement Using Level Numbers

The following topics are covered:

- Structuring and Grouping Your Definitions
- Level Numbers in View Definitions
- Level Numbers in Field Groups
- Level Numbers in Redefinitions

Structuring and Grouping Your Definitions

Level numbers are used within the DEFINE DATA statement to indicate the structure and grouping of the definitions. This is relevant with:

- view definitions

- field groups
- redefinitions

Level numbers are 1- or 2-digit numbers in the range from 01 to 99 (the leading zero is optional).

Generally, variable definitions are on Level 1.

The level numbering in view definitions, redefinitions and groups must be sequential; no level numbers may be skipped.

Level Numbers in View Definitions

If you define a view, the specification of the view name must be on Level 1, and the fields the view is comprised of must be on Level 2. (For details on view definitions, see *Database Access*.)

Example of Level Numbers in View Definition:

```
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF EMPLOYEES
  2 NAME
  2 FIRST-NAME
  2 BIRTH
...
END-DEFINE
```

Level Numbers in Field Groups

The definition of groups provides a convenient way of referencing a series of consecutive fields. If you define several fields under a common group name, you can reference the fields later in the program by specifying only the group name instead of the names of the individual fields.

The group name must be specified on Level 1, and the fields contained in the group must be one level lower.

For group names, the same naming conventions apply as for user-defined variables.

Example of Level Numbers in Group:

```
DEFINE DATA LOCAL
1 #FIELDA (N2.2)
1 #FIELDB (I4)
1 #GROUPA
  2 #FIELDC (A20)
  2 #FIELDD (A10)
  2 #FIELDE (N3.2)
1 #FIELDF (A2)
...
END-DEFINE
```

In this example, the fields #FIELDC, #FIELDD and #FIELDE are defined under the common group name #GROUPA. The other three fields are not part of the group. Note that #GROUPA only serves as a group name and is not a field in its own right (and therefore does not have a format/length definition).

Level Numbers in Redefinitions

If you redefine a field, the REDEFINE option must be on the same level as the original field, and the fields resulting from the redefinition must be one level lower. For details on redefinitions, see *Redefining Fields*.

Example of Level Numbers in Redefinition:

```
DEFINE DATA LOCAL
1 VIEWEMP VIEW OF STAFFDDM
  2 BIRTH
  2 REDEFINE BIRTH
    3 #YEAR-OF-BIRTH  (N4)
    3 #MONTH-OF-BIRTH (N2)
    3 #DAY-OF-BIRTH   (N2)
1 #FIELDA (A20)
1 REDEFINE #FIELDA
  2 #SUBFIELD1 (N5)
  2 #SUBFIELD2 (A10)
  2 #SUBFIELD3 (N5)
...
END-DEFINE
```

In this example, the database field BIRTH is redefined as three user-defined variables, and the user-defined variable #FIELDA is redefined as three other user-defined variables.

Storage Alignment

The storage area, in which all user-defined variables are stored, always begins on a double-word boundary.

If a DEFINE DATA statement is used, all data blocks (for example, LOCAL, GLOBAL blocks) are double-word aligned, and all hierarchical structures (view definitions and groups) on Level 1 are full-word aligned. Redefinitions, scalar and array variables are not aligned, even if they are defined at level 1.

Alignment within the data area is the responsibility of the user and is governed by the order in which variables are defined to Natural.