

ON ERROR

Structured Mode Syntax

```

ON ERROR
  statement ...
END-ERROR

```

Reporting Mode Syntax

```

ON ERROR { statement ...
          DO statement ... DOEND }

```

This chapter covers the following topics:

- Function
- Restriction
- Syntax Description
- ON ERROR Processing within Objects on Different Levels
- System Variables
- Example

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

Related Statements: DECIDE FOR | DECIDE ON | IF | IF SELECTION

Function

The ON ERROR statement is used to intercept execution time errors which would otherwise result in a Natural error message, followed by termination of Natural program execution, and a return to command input mode.

When the ON ERROR statement block is entered for execution, the normal flow of program execution has been interrupted and cannot be resumed except for Natural error 3145 (record requested in hold), in which case a RETRY statement will cause processing to be resumed exactly where it was suspended.

This statement is non-procedural (that is, its execution depends on an event, not on where in a program it is located).

Restriction

Only one ON ERROR statement is permitted in a Natural object.

Syntax Description

Syntax Element	Description
<i>statement . . .</i>	<p>Defining the ON ERROR Processing:</p> <p>To define the processing that shall take place when an ON ERROR condition has been encountered, you can specify one or multiple statements.</p> <p>Exiting from an ON ERROR Block:</p> <p>An ON ERROR block may be exited by using a FETCH, STOP, TERMINATE, RETRY or ESCAPE ROUTINE statement. If the block is not exited using one of these statements, standard error message processing is performed and program execution is terminated.</p>
END-ERROR	<p>End of ON ERROR Statement Block:</p> <p>The Natural reserved word END-ERROR must be used to end an ON ERROR statement block.</p>

ON ERROR Processing within Objects on Different Levels

In an object call hierarchy created by means of CALLNAT, PERFORM or FETCH RETURN statements, each object may contain an ON ERROR statement.

When an error occurs, Natural will trace back the call hierarchy and select the first ON ERROR statement encountered in an object for execution.

For further information, see *Processing of Application Errors* in the *Programming Guide*.

System Variables

The following Natural system variables can be used in conjunction with the ON ERROR statement (as shown in the Example below):

System Variable	Explanation
*ERROR-NR	Contains the number of the error detected by Natural.
*ERROR-LINE	Contains the line number of the statement which caused the error.
*PROGRAM	Contains the name of the Natural object that is currently being executed.

Example

```
** Example 'ONEEX1': ON ERROR
```

```
**
```

```
**
```

```
CAUTION: Executing this example will modify the database records!
```

```
*****
```

```
DEFINE DATA LOCAL
```

```
1 EMPLOY-VIEW VIEW OF EMPLOYEES
```

```
2 NAME
```

```
2 CITY
```

```
*
```

```
1 #NAME (A20)
```

```
1 #CITY (A20)
```

```
END-DEFINE
```

```
*
```

```
REPEAT
```

```
INPUT 'ENTER NAME:' #NAME
```

```
IF #NAME = ' '
```

```
STOP
```

```
END-IF
```

```
FIND EMPLOY-VIEW WITH NAME = #NAME
```

```
INPUT (AD=M) 'ENTER NEW VALUES:' ///
```

```
'NAME:' NAME /
```

```
'CITY:' CITY
```

```
UPDATE
```

```
END TRANSACTION
```

```
/*
```

```
ON ERROR
```

```
IF *ERROR-NR = 3009
```

```
WRITE 'LAST TRANSACTION NOT SUCCESSFUL'
```

```
/ 'HIT ENTER TO RESTART PROGRAM'
```

```
FETCH 'ONEEX1'
```

```
END-IF
```

```
WRITE 'ERROR' *ERROR-NR 'OCCURRED IN PROGRAM' *PROGRAM
```

```
'AT LINE' *ERROR-LINE
```

```
FETCH 'MENU'
```

```
END-ERROR
```

```
/*
```

```
END-FIND
```

```
END-REPEAT
```

```
END
```