

Error State Handler ON ERROR

Error state handler ON ERROR

An error handler method may be controlled by using these actions:

- [ON ERROR](#)
- [RESUME](#)
- [RETRY](#)

ON ERROR action sets a behaviour of the script interpreter when an error occurs, as follows:

ON ERROR NONE - sets the default error handler (termination of the script execution).

ON ERROR *Label* - setting of an error handler. If an error occurs, the script will not be terminated, but the interpreter will jump to the given label. The predefined local variables `_ERR_NR`, `_ERR_MSG` and `_ERR_LINE` automatically get the error description before the jump. Within the error handler, it is allowed to perform the actions [RETRY](#) (repeated execution of the action that caused the error) or [RESUME](#) (the script proceeds in execution after the action that caused the error). The label must be placed in the same procedure as the action (or in the initialization part).

ON ERROR OFF - if an error occurs, the processing sequence of actions is not interrupted and the local variables `_ERR_NR`, `_ERR_MSG` and `_ERR_LINE` are adjusted. Testing `of _ERR_NR` may detect and handle the error. In this mode, the access to the variable `_ERR_NR` (value reading) will set this variable to `_ERR_NO_ERROR` value. The object will be reopened.

ON ERROR ON - recovery of the error handling state that was valid before **ON ERROR OFF** action.

Note:

A change of error behaviour setting is valid only within the particular code section ([procedure](#),...). If the action is used in a procedure, changes in the setting are cancelled after its termination and set to the state defined before the procedure was [called](#).

If an error occurs in a procedure, there are the possibilities, according to the error handling setting:

- If a procedure defines the error handling **ON ERROR**, then the error is to be processed according to the description above
- If a procedure does not define the error handling, then the procedure is terminated and the error behaves as if it occurred in the action of calling a procedure:

```
PROCEDURE Proc
  IF 1
    THEN ; error occurs here
  RETURN
ENDIF
END Proc
```

```
BEGIN
  CALL Proc
END
```

Event written in this way will be terminated with an error.

```
PROCEDURE Proc
  IF 1 THEN ; error occurs here
  RETURN
ENDIF
END Proc
```

```
BEGINN
  ON ERROR ErrorHandler
```

```
  CALL Proc
  END
  ErrorHandler:
  END
```

Error that occurs in the procedure **Proc** will terminate it. Since the error is not handled, the error acts as if it occurred during the execution of the action **CALL Proc** and thus **ErrorHandler** will be called.

```
PROCEDURE Proc
ON ERROR ErrorHandlerProc
IF 1 THEN ; error occurs here
RETURN
ENDIF
RETURN
```

```
ErrorHandlerProc:
RETURN
END Proc
```

```
BEGIN
ON ERROR ErrorHandler
CALL Proc
END
ErrorHandler:
END
```

An error that occurs in **Proc** procedure will cause jumping to **ErrorHandlerProc**. Since the error is handled, the procedure will be normally terminated and **ErrorHandler** will not be called.

Error status is described by line number where the error occurred (from the view of ESL script that forms the error description), error code ([Error codes](#)) and detail error description in some cases.

Above mentioned is followed by complete call dump (CALL action) by which the action (ended by error) was performed. Detail description of call dump is mentioned in description of the function [%GetCallChain](#).



Related pages:

[Event Script Language \(ESL\)](#)
[Error state handler](#)