# CALL FILE

### Structured Mode Syntax

```
CALL FILE 'program-name' operand1  operand2
  statement ...
END-FILE
```

### Reporting Mode Syntax

```
CALL FILE 'program-name' operand1  operand2
  statement ...
[LOOP]
```

This chapter covers the following topics:

- Function
- Restriction
- Syntax Description
- Example

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

Related Statements: CALL | CALL LOOP | CALLNAT | DEFINE SUBROUTINE | ESCAPE | FETCH | PERFORM

Belongs to Function Group: *Invoking Programs and Routines*

---

## Function

The `CALL FILE` statement is used to call a non-Natural program which reads a record from a non-Adabas file and returns the record to the Natural program for processing.

## Restriction

The statements `AT BREAK`, `AT START OF DATA` and `AT END OF DATA` must not be used within a `CALL FILE` processing loop.

# Syntax Description

Operand Definition Table:

| Operand | Possible Structure | | | | | Possible Formats | | | | | | | | | | | | Referencing Permitted | Dynamic Definition |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| *operand1* | | S | A | | | A | U | N | P | I | F | B | D | T | L | C | | | yes | yes |
| *operand2* | | S | A | G | | A | U | N | P | I | F | B | D | T | L | C | | | yes | yes |

Syntax Element Description:

| | |
|---|---|
| ***'program-name'*** | The name of the non-Natural program to be called. |
| ***operand1*** | Control field: *operand1* is used to provide control information. |
| ***operand2*** | *operand2* defines the record area.<br><br>The format of the record to be read can be described using field definitions (or FILLER *n*X) entries following the name of the first field in the record. The fields used to define the record format must not have been previously defined in the Natural program. This ensures that fields are allocated in the contiguous storage by Natural. |
| ***statement ...*** | The `CALL FILE` statement initiates a processing loop which must be terminated with an `ESCAPE` or `STOP` statement. More than one `ESCAPE` statement may be specified to escape from a `CALL FILE` loop based on different conditions. |
| **END-FILE** | An `END-FILE` statement must be used to close the processing loop. |

# Example

### Calling Program:

```
** Example 'CFIEX1': CALL FILE
************************************************************************
DEFINE DATA LOCAL
1 #CONTROL (A3)
1 #RECORD
  2 #A     (A10)
  2 #B     (N3.2)
  2 #FILL1 (A3)
  2 #C     (P3.1)
END-DEFINE
*
CALL FILE 'USER1' #CONTROL #RECORD
  IF #CONTROL = 'END'
    ESCAPE BOTTOM
  END-IF
END-FILE
/*****************************
/* ... PROCESS RECORD ...
/*****************************
END
```

The byte layout of the record passed by the called program to the Natural program in the above example is as follows:

```
CONTROL #A       #B   FILLER  #C
(A3)   (A10)   (N3.2)  3X   (P3.1)

xxx xxxxxxxxxx xxxxx   xxx   xxx
```

## Called COBOL Program:

```
ID DIVISION.
PROGRAM-ID. USER1.
ENVIRONMENT DIVISION.
INPUT-OUTPUT SECTION.
FILE-CONTROL.
     SELECT USRFILE ASSIGN UT-S-FILEUSR.
DATA DIVISION.
FILE SECTION.
FD   USRFILE RECORDING F LABEL RECORD OMITTED
     DATA RECORD DATA-IN.
01   DATA-IN         PIC X(80).
LINKAGE SECTION.
01   CONTROL-FIELD   PIC XXX.
01   RECORD-IN       PIC X(21).
PROCEDURE DIVISION USING CONTROL-FIELD RECORD-IN.
BEGIN.
     GO TO FILE-OPEN.
FILE-OPEN.
     OPEN INPUT USRFILE
     MOVE SPACES TO CONTROL-FIELD.
     ALTER BEGIN TO PROCEED TO FILE-READ.
FILE-READ.
     READ USRFILE INTO RECORD-IN
          AT END
          MOVE 'END' TO CONTROL-FIELD
          CLOSE USRFILE
          ALTER BEGIN TO PROCEED TO FILE-OPEN.
     GOBACK.
```