

# 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 ]
```

Dieses Kapitel behandelt folgende Themen:

- Funktion
- Einschränkung
- Syntax-Beschreibung
- Beispiel

Eine Erläuterung der in dem Syntax-Diagramm verwendeten Symbole entnehmen Sie dem Abschnitt *Syntax-Symbole*.

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

Gehört zur Funktionsgruppe: *Aufrufen von Programmen und Unterprogrammen*

---

## Funktion

Das Statement CALL FILE dient dazu, ein nicht in Natural geschriebenes Programm aufzurufen, das einen Datensatz von einer Nicht-Adabas-Datei liest und diesen Datensatz an das aufrufende Natural-Programm zur Verarbeitung übergibt.

## Einschränkung

Innerhalb einer CALL FILE-Schleife dürfen die Statements AT BREAK, AT START OF DATA und AT END OF DATA nicht verwendet werden.

# Syntax-Beschreibung

Operanden-Definitionstabelle:

Operand	Mögliche Struktur	Mögliche Formate	Referenzierung erlaubt	Dynam. Definition
<i>operand1</i>	S A	A U N P I F B D T L C	ja	ja
<i>operand2</i>	S A G	A U N P I F B D T L C	ja	ja

Syntax-Element-Beschreibung:

<b>'<i>program-name</i>'</b>	Der Name des aufzurufenden Nicht-Natural-Programmes.
<b><i>operand1</i></b>	Kontrollfeld: <i>operand1</i> dient dazu, Kontrollinformationen zu liefern.
<b><i>operand2</i></b>	<i>operand2</i> definiert den Datensatz-Bereich.  Das Format des zu lesenden Datensatzes kann mit Felddefinitionseinträgen (oder FILLER <i>nX</i> ), die hinter dem ersten Feld des Datensatzes stehen, beschrieben werden. Die Felder, die dazu dienen, das Format des Datensatzes zu definieren, brauchen im Natural-Programm nicht vorher definiert werden. Dadurch ist gewährleistet, dass Natural die Felder benachbarten Speicherplätzen zuordnet.
<b><i>statement ...</i></b>	Das Statement CALL FILE initiiert eine Verarbeitungsschleife, die mit einem ESCAPE- oder STOP-Statement beendet werden muss. Um die Schleife in Abhängigkeit von verschiedenen Bedingungen zu beenden, können Sie mehrere ESCAPE-Statements verwenden.
<b>END-FILE</b>	Ein END-FILE-Statement muss benutzt werden, um die Verarbeitungsschleife zu schließen.

## Beispiel

**Aufrufendes Programm:**

```

** 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

```

Die Byte-Belegung des vom aufgerufenen Programm an das Natural-Programm übergebenen Datensatzes sieht folgendermaßen aus:

```

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

xxx xxxxxxxxxxxx xxxxx   xxx   xxx

```

### Aufgerufenes COBOL-Programm:

```

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.

```