Chapter 8

Input/Output Statements

This chapter contains the following subsections:

- "Statement Summary"
- "ACCEPT"
- "BACKSPACE"
- "CLOSE"
- "DECODE"
- "DEFINE FILE"
- "DELETE"
- "ENCODE"
- "ENDFILE"
- "FIND"
- "INQUIRE"
- "OPEN"
- "PRINT or TYPE"
- "READ (Direct Access)"
- "READ (Indexed)"
- "READ (Internal)"
- "READ (Sequential)"
- "REWIND"
- "REWRITE"
- "UNLOCK"
- "WRITE (Direct Access)"
- "WRITE (Indexed)"
- "WRITE (Internal)"
- "WRITE (Sequential)"
- "Control Information List — cilist"
- "Input/Output List — iolist"
- "Data Transfer Rules"

This chapter describes the statements that control the transfer of data within internal storage and between internal and external storage devices. It provides an overview of the Fortran I/O statements and gives syntax, rules, and examples for each.

This chapter also describes general rules that apply to data transfer statements.

Statement Summary

The I/O statements described in this chapter are grouped into the following classes:

- Data transfer statements, which transfer information between two areas of internal storage or between internal storage and an external file. The seven types are
  - **READ**
  - **DELETE**
  - **UNLOCK**
  - **ACCEPT**
  - **WRITE**
  - **REWRITE**
  - **PRINT** or **TYPE**
- Auxiliary statements, which explicitly open or close a file, provide current status information about a file or unit or write an endfile record. The four types are
  - **OPEN**
  - **CLOSE**
  - **INQUIRE**
  - **ENDFILE**
- File positioning statements, which position data files to the previous record or to the initial point of a file. These statements apply only to external files. They are
  - **BACKSPACE**
  - **REWIND**
- Statements that provide compatibility with earlier versions of Fortran. They are included to permit the older Fortran programs to be compiled and exist on the same system as standard Fortran 77 programs. The statements include the following:
  - **ENCODE**
  - **DECODE**
  - **DEFINE FILE**
  - **FIND**

The following sections describe the statements in the above summary in detail.

ACCEPT

The **ACCEPT** statement transfers data from the standard input unit to the items specified by the input list.

Syntax

ACCEPT *f* [,*iolist*]

where

f is the format specifier

iolist is an optional output list specifying where the data is to be stored.

See "Control Information List — cilist" and "Input/Output List — iolist" for a description of the *f* and *iolist* parameters.

Rules for Use

The **ACCEPT** statement specifies formatted input from the file associated with the system input unit; it cannot be connected to a user–specified input unit.

See "Data Transfer Rules" for additional rules.

Example

The following code transfers character data from the standard input unit into **x**.

```
      ACCEPT 3,x
 3    FORMAT (A)
```

BACKSPACE

The **BACKSPACE** statement positions a data file before the preceding record. It can be used with both formatted and unformatted data files.

Syntax

BACKSPACE *u*
BACKSPACE (*alist*)

where

u is an external unit identifier.

alist is a list of the following specifiers:

1.
   [UNIT =] *u* is a required unit specifier. *u* must be an integer expression that identifies the number of an external unit. If the keyword **UNIT =** is omitted, then *u* must be the first specifier in *alist*.

IOSTAT = *ios* is an I/O status *specifier* that specifies the variable to be defined with a status value by the **BACKSPACE** statement. A zero value for *ios* denotes a no error condition, while a positive integer value denotes an error condition.

ERR = *s* is an error specifier that identifies a statement number to which control is transferred when an error condition occurs during the execution of the **BACKSPACE** statement.

Note: An error message is issued if this statement references a file opened with an **ACCESS="KEYED"**, **ACCESS="APPEND"**, or a **FORM="SYSTEM"** specification.

Method of Operation

The unit specifier is required and must appear exactly once. The other specifiers are optional and can appear at most once each in the *alist*. Specifiers can appear in any order. For information about exceptions refer to "Unit Specifier— UNIT".

The **BACKSPACE** statement positions the file on the preceding record. If there is no preceding record, the position of the file is unchanged. If the preceding record is an endfile record, the file is positioned before the endfile record.

Examples

```
BACKSPACE M
BACKSPACE (6, IOSTAT=LP, ERR=998)
```

CLOSE

The **CLOSE** statement disconnects a particular file from a unit.

Syntax

`CLOSE` (*cilist*)

where *cilist* is a list of the following specifiers:

[UNIT =] *u* is a required unit specifier. *u* must be an integer expression that identifies the number of an external unit. If the keyword **UNIT=** i*s* omitted, then *u* must be the first specifier in *cilist.*

IOSTAT=*ios* is an I/O status specifier that specifies the variable to be defined with a status value by the **CLOSE** statement. A zero value for *ios* denotes a no error condition while a positive integer value denotes an error condition.

DISP[**OSE**]=*disposition*

Provides the same function as the like parameters in the **OPEN** statement. The *disposition* parameters in the file's **CLOSE** statement override the disposition parameters in its **OPEN** statement.

ERR=*s* is an error specifier that identifies a statement number to which control is to be transferred when an error condition occurs during execution of the **CLOSE** statement.

STATUS='*sta*' is a file status specifier. *sta* is a character expression that, when any trailing blanks are removed, has a value of **KEEP** or **DELETE**. The status specifier determines the disposition of the file that is connected to the specified unit. **KEEP** specifies that the file is to be retained after the unit is closed. **DELETE** specifies that the file is to be deleted after the unit is closed. If a file has been opened for **SCRATCH** in an **OPEN** statement, then **KEEP** must not be specified in the **CLOSE** statement. If *iolist* contains no file status specifier, the default value is **KEEP**, except when the file has been opened for **SCRATCH**, in which case the default is **DELETE**.

Method of Operation

At the normal termination of an executable program, all units that are connected are closed. Each unit is closed with status **KEEP** unless the file has been opened for **SCRATCH** in an **OPEN** statement. In the latter case, the unit is closed as if with file status **DELETE**.

A **CLOSE** statement need not occur in the same program unit in which the file was opened. A **CLOSE** statement that specifies a unit that does not exist or has no file connected to it does not affect any file, and is permitted.

A unit that is disconnected by a **CLOSE** statement can be reconnected within the same executable program, either to the same file or to a different file. A file that is disconnected can be reconnected to the same unit or a different unit, provided that the file still exists.

Examples

```
CLOSE(UNIT=1,STATUS='KEEP')
CLOSE(UNIT=K,ERR=19,STATUS='DELETE')
```

DECODE

The **DECODE** statement transfers data between internal files, decoding the transferred data from character format to internal format.

Note: This statement provides primarily the same function as the **READ** statement using internal files, except that the input is read from a numeric scalar or array rather than a character string. This release does not support the concept of multiple records, and you must specify the record length. Where possible, use a **READ** statement instead of **DECODE** in new programs to make them compatible with different Fortran 77 operating environments.

Syntax

`DECODE (`*n*`,`*f*`,`*target*`[,ERR=`*s*`][,IOSTAT=`*rn*`]) [`*iolist*`]`

where

n is an integer expression specifying the number of characters to be translated to internal format.

f	is a format specifier (as described in "Format Specifier — FMT" in this chapter).
target	is a scalar reference or array indicating the destination of the characters after translation to external form.
ERR=*s*	See "Control Information List — cilist" in this chapter for an explanation of this parameter.
IOSTAT=*rn*	See "Control Information List — cilist" in this chapter for an explanation of this parameter.
iolist	is an optional list specifying the source data, as described in "Input/Output List — iolist" of this chapter.

Method of Operation

- The relationship between the I/O list and the format specifier is the same as for formatted I/O.
- The maximum number of characters transmitted is the maximum number possible for the *target* data type. If *target* is an array, the elements are processed in subscript order.

DEFINE FILE

The **DEFINE FILE** statement defines the size and structure of a relative file and connects it to a unit. It primarily provides the same function as the Fortran **OPEN** statement specifying **ACCESS='DIRECT'**.

Syntax

```
DEFINE FILE u (reccount,reclen,U,asvar)[,u (reccount,reclen,U,asvar)] ...
```

where

u	is an integer expression that identifies the number of an external unit that contains the file.
reccount	is an integer expression defining the number of records in the file.
reclen	is an integer expression specifying in word (two byte) units the length of each record.
U	specifies an unformatted (binary) file. *U* is always required and always in the same position, as shown in the above syntax.
asvar	is an associated integer variable indicating the next higher numbered record to be read or written. It is updated after each direct−access I/O operation.

Method of Operation

Only unformatted files can be opened with a **DEFINE FILE** statement. The file defined by *u* is assumed to contain fixed−length records of *recln* (two byte) words each. The records in the file are numbered 1 through *reccount*. The **DEFINE FILE** statement or equivalent **OPEN** statement must be executed before executing a **READ**, **WRITE**, or other direct–access statement. The first direct–access **READ** for the specified file opens an existing file; if the file does not exist, an error condition occurs. The first direct–access **WRITE** for the specified file opens the file and creates a new relative file.

DELETE

The **DELETE** statement removes a record from an indexed file. An error condition occurs if the file is not indexed.

Syntax

`DELETE [UNIT=]`*unum*

or

`DELETE ([UNIT=]`*unum*`[,IOSTAT=`*rn*`][,ERR=`*s*`]`

where

UNIT=*unum*	is a unit or internal file to be acted on.
IOSTAT=*rn*	is the name of variable in which I/O completion status is posted.
ERR=s	is a statement label to which control is transferred after an error.

See "Control Information List — cilist" and "Input/Output List — iolist" for details on these parameters.

Method of Operation

The **DELETE** statement deletes the current record, which is the last record accessed on unit *u*.

Example

The following statement deletes the last record read in from the file connected to logical unit 10.

```
DELETE (10)
```

ENCODE

The **ENCODE** statement transfers data between internal files, encoding the transferred data from internal format to character format.

Note: This statement primarily provides the same function as the **WRITE** statement, using internal files. Except that the input is read from a numeric scalar or array rather than a character string, the concept of multiple records is not supported. The record length is user specified. Where possible, use a **WRITE** statement instead of **ENCODE** in new programs to make them compatible with different Fortran 77 operating environments.

Syntax

`ENCODE (`*n*`,`*f*`,`*target*`[,ERR=`*s*`][,IOSTAT=`*rn*`]) [`*iolist*`]` where

n is an integer expression specifying the number of characters to be translated to character format.

f is a format specifier (as described in the "Format Specifier — FMT").

ERR=*s* See "Control Information List — cilist" for an explanation of this parameter.

IOSTAT=*rn* See "Control Information List — cilist" for an explanation of this parameter.

target is a scalar reference or array indicating the destination of the characters after translation to external form.

iolist is an optional list specifying the source data, as described in "Input/Output List — iolist".

Method of Operation

The relationship between the I/O list and the format specifier is the same as for formatted I/O. *target* is padded with blanks if fewer than *n* characters are transferred. The maximum number of characters transmitted is the maximum number possible for the *target* data type. If *target* is an array, the elements are processed in subscript order.

ENDFILE

The **ENDFILE** statement writes an endfile record as the next record of the file. It can be used with both unformatted and formatted data files.

Syntax

```
ENDFILE u
ENDFILE (alist)
```

where

u is an external unit identifier

alist is a list of the following specifiers:

[UNIT =]*u* is a required unit specifier. *u* must be an integer expression that identifies the number of an external unit. If the keyword **UNIT=** is omitted, then *u* must be the first specifier in *alist*.

IOSTAT=*ios* is an I/O status specifier that specifies the variable to be defined with a status value by the **ENDFILE** statement. A zero value for *ios* denotes a no error condition, while a positive integer value denotes an error condition.

ERR=*s* is an error specifier that identifies a statement number to which control is transferred when an error condition occurs during the execution of the **ENDFILE** statement.

Note: An error message is issued if this statement references a keyed–access file.

Method of Operation

The unit specifier is required and must appear exactly once. The other specifiers are optional and can appear at most once each in the *alist*. Specifiers can appear in any order (for exceptions refer to "Unit Specifier — UNIT").

An **ENDFILE** statement writes an endfile record. The specified file is then positioned after the endfile record. If a file is connected for direct access, only those records before the endfile record are considered to have been written and thus can be read in subsequent direct–access connections to the file.

An **ENDFILE** statement for a file that is connected but does not exist creates the file.

After an **ENDFILE** statement, a **BACKSPACE** or **REWIND** statement must be used to reposition the file before the execution of any data transfer I/O statement.

Note: If the program is compiled with the **-vms_endfile** option, the file can still be written to after the endfile record.

Examples

The following statements are examples of **ENDFILE** statements.

```
ENDFILE 2
ENDFILE (2,IOSTAT=IE, ERR=1000)
```

FIND

The **FIND** statement positions a file to a specified record number and sets the associate variable number (defined in an **OPEN** or **DEFINE FILE** statement) to reflect the new position. It is functionally equivalent to a direct–access **READ** statement except that no *iolist* is specified and no data transfer takes place. The statement opens the file if it is not already open.

Syntax

FIND ([UNIT=]*u*,REC=*rn*[,ERR=*s*][,IOSTAT=*rn*])

where

u — is an integer expression that identifies the number of an external unit that contains the file. The number must refer to a relative file.

ERR=*s*, **IOSTAT=***rn*, **REC=***rn* — See "Control Information List — cilist" for an explanation of these parameters.

INQUIRE

The **INQUIRE** statement inquires about the properties of a particular named file or the file connected to a particular unit. There are two forms: inquire by file and inquire by unit.

Syntax

```
INQUIRE (FILE=fname, [DEFAULTFILE=fname ...,]inqlist)
INQUIRE ([UNIT=]u,inqlist)
```

where

FILE=*fname* is a file specifier. *fname* is a character expression that specifies the name of the file being queried. The named file need not exist or be connected to a unit.

DEFAULTFILE=*fname*

This parameter corresponds to the **DEFAULTFILE** parameter in an **OPEN** statement and is used to inquire about a file assigned a default name when it was opened. See "OPEN" for details.

[UNIT=]*u* is a unit specifier. *u* must be an integer expression that identifies the number of an external unit. The specified unit need not exist or be connected to a file. If the keyword **UNIT=** is omitted, then *u* must be the first specifier in *inqlist*.

inqlist is composed of one or more of the following specifiers, separated by commas:

ACCESS=*acc* *acc* is a character variable or character array element to be assigned a value by the **INQUIRE** statement. The value assigned describes the type of file access as shown in Table 8−1

Table 8−1 File Access Types

Value Assigned	File Access
SEQUENTIAL	Sequential
DIRECT	Direct
KEYED	Keyed
UNKNOWN	No connection

BLANK=*blnk* *blnk* is a character variable or character array element to be assigned a value by the **INQUIRE** statement. The value assigned describes the blank specifier for the file as shown in Table 8−2

Table 8−2 Blank Control Specifiers

Value of *blnk*	Specifier
NULL	Null blank control, connected for formatted I/O
ZERO	Zero blank control
UNKNOWN	Not connected or not connected for formatted I/O

CARRIAGECONTROL=*ccspec*

ccspec is assigned one of the following carriage control specifications made in the **OPEN** statement for the file: **FORTRAN**, **LIST**, **NONE**, or **UNKNOWN**.

DIRECT=*dir* *dir* is a character variable or character array element to be assigned a value by the **INQUIRE** statement. *dir* is assigned the value **YES** if **DIRECT** is a legal access method for the file; it is assigned the value **NO** if **DIRECT** is not a legal access method. If the processor is unable to determine the access type, *dir* is assigned the value **UNKNOWN**.

ERR=*s* is an error specifier that identifies a statement number to which control is transferred when an error condition occurs during the execution of the **INQUIRE** statement.

EXIST=*ex* *ex* is a logical variable or logical array element to be assigned a value by the **INQUIRE** statement. *ex* is assigned the value **.TRUE.** if the specified unit or file exists; otherwise, *ex* is assigned the value **.FALSE.** . A unit exists if it is a number in the range allowed by the processor.

FORM=*fm* *fm* is a character variable or character array element to be assigned a value by the **INQUIRE** statement. The value assigned is the form specifier for the file as shown in Table 8−3

Table 8−3 Form Specifiers

Value for *fm*	Specifier
FORMATTED	Formatted I/O
UNFORMATTED	Unformatted I/O
UNKNOWN	Unit is not connected

FORMATTED=*fmt*

fmt is a character variable or character array element to be assigned a value by the **INQUIRE** statement. *fmt* is assigned the value **YES** if **FORMATTED** is a legal form for the file; *fmt* is assigned the value **NO** if **FORMATTED** is not a legal form. If the processor is unable to determine the legal forms of data transfer, *fmt* is assigned the value **UNKNOWN**.

IOSTAT=*ios* is an I/O status specifier that specifies the variable to be defined with a status value by the **INQUIRE** statement. A zero value for *ios* denotes a no error condition, while a positive integer value denotes an error condition.

KEYED=*keystat*

keystat is a character scalar memory reference assigned a value as shown in Table 8−4

Table 8−4 Keyed−Access Status Specifiers

keystat	Meaning
YES	Indexed file, keyed access allowed
NO	Keyed access not allowed
UNKNOWN	Access type undetermined

NAMED=*nmd* *nmd* is a logical variable or logical array element to be assigned a value by the **INQUIRE** statement. *nmd* is assigned the value **.TRUE.** if the file has a name. Otherwise, *nmd* is assigned the value **.FALSE.**.

NAME=*fn* *fn* is a character variable or character array element to be assigned a value by the **INQUIRE** statement. *fn* is assigned the name of the file if the file has a name. Otherwise, *fn* is undefined. If the **NAME** specifier appears in an **INQUIRE** by file statement, its value is not necessarily the same as the name given in the file specifier.

NEXTREC=*nr* *nr* is an integer variable or integer array element to be assigned a value by the

INQUIRE statement. *nr* is assigned the value *n* + 1, where *n* is the record number of the last record read or written for direct access on the specified unit or file. If the file is connected but no records have been read or written, *nr* is assigned the value 1. If the file is not connected for direct access, *nr* is assigned the value 0.

NUMBER=*num* — *num* is an integer variable or integer array element that is assigned a value by the **INQUIRE** statement. *num* is assigned the external unit identifier of the unit currently connected to the file. *num* is undefined if there is no unit connected to the file. This specifier cannot be used with an **INQUIRE** by unit statement (**INQUIRE** (*iulist*)).

OPENED=*od* — *od* is a logical variable or logical array element to be assigned a value by the **INQUIRE** statement. *od* is assigned the value **.TRUE.** if the file specified is connected to a unit or if the specified unit is connected to a file. Otherwise, *od* is assigned the value **.FALSE.**.

ORGANIZATION=*org*

org is a character scalar memory reference assigned the value of the file organization established when the file was opened; it has one of the following values: **SEQUENTIAL**, **RELATIVE**, **INDEXED**, or **UNKNOWN** (always assigned to unopened files).

RECL=*rcl* — *rcl* is an integer variable or integer array element to be assigned a value by the **INQUIRE** statement. *rcl* is assigned the value of the record length in number of characters for formatted files and in words for unformatted files. If there is no connection or if the connection is not for direct access, *rcl* becomes undefined.

RECORDTYPE=*rectype*

rectype is a character scalar memory reference assigned the value of the record type file established when the file was opened; it has one of the following values:**FIXED**, **VARIABLE**, **STREAM_LF**, or **UNKNOWN**.

SEQUENTIAL=*seq*

seq is a character variable or character array element to be assigned a value by the **INQUIRE** statement. *seq* is assigned the value **YES** if **SEQUENTIAL** is a legal access method for the file. **seq** is assigned the value **NO** if **SEQUENTIAL** is not a legal access method. If the processor is unable to determine the legal access methods, *seq* is assigned the value **UNKNOWN**.

UNFORMATTED=*unf*

unf is a character variable or character array element to be assigned a value by the **INQUIRE** statement. *unf* is assigned the value of **YES** if **UNFORMATTED** is a legal format for the file; *unf* is assigned the value **NO** if **UNFORMATTED** is not a legal format for the file. If the processor is unable to determine the legal form,*unf* is assigned the value **UNKNOWN**.

Method of Operation

Specifiers can be given in *iflist* or *iulist* in any order ("Unit Specifier— UNIT" lists exceptions).

An **INQUIRE** statement assigns values to the specifier variables or array elements *nmd*, *fn*, *seq*, *dir*, *fmt*, and *unf* only if the value of the file specifier *fname* is accepted by the processor and if a file exists by that name. Otherwise, these specifier variables become undefined. Each specifier can appear at most once in the *iflist* or *iulist*, and the list must contain at least one specifier.

An **INQUIRE** statement assigns values to the specifier variables or array elements *num*, *nmd*, *fn*, *acc*, *seq*, *dir*, *fm*, *fmt*, *unf*, *rcl*, *nr*, and *blnk* only if the specified unit exists and if a file is connected to it. Otherwise, these specifier variables become undefined. However, the specifier variables *ex* and *od* are always defined unless an error condition occurs. All inquiry specifier variables except *ios* become undefined if an error condition occurs during execution of an **INQUIRE** statement.

Examples

The following examples show **INQUIRE** statements.

```
INQUIRE (FILE='MYFILE.DATA',NUMBER=IU,RECL=IR)
INQUIRE (UNIT=6, NAME=FNAME)
```

OPEN

The **OPEN** statement creates files and connects them to units. It can create a preconnected file, create and connect a file, connect an existing file, or reconnect an already connected file. See "File Positions" in Chapter 1 of the Fortran 77 Programmer's Guide for information on the relative record position in a file after an **OPEN** is executed.

Syntax

`OPEN (`*olist*`)`

where *olist* is a list of the following specifiers, separated by commas:

[**UNIT=**] *u* — is a required unit specifier. *u* must be an integer expression that identifies the number of an external unit. If the keyword **UNIT=** is omitted, then the *u* must be the first specifier in *olist.*

IOSTAT=*ios* — is an I/O status specifier that identifies the variable to be defined with a status value by the **OPEN** statement. A zero value for *ios* denotes a no error condition, while a positive integer value denotes an error condition.

ERR=s — is an error specifier that identifies a statement number to which program control is to be transferred when an error condition occurs during execution of the **OPEN** statement.

FILE=*fname* — is a file specifier. *fname* is a character expression specifying the name of the external file to be connected. The file name must be a name allowed by the processor. *fname* can also be a numeric variable to which Hollerith data is assigned. A null character terminates the filename. Three VMS predefined system logical names—**SYS$INPUT**, **SYS$OUTPUT**, and **SYS$ERROR**—are supported. These names allow an **OPEN**

statement to associate an arbitrary unit number to standard input, standard output, and standard error, respectively, instead of the standard predefined logical unit numbers 5, 6, and 0.

ACCESS=*acc* is an access specifier. *acc* is a character expression that, when trailing blanks are removed, has one of the following values: **SEQUENTIAL**, **DIRECT**, **KEYED**, or **APPEND**.**SEQUENTIAL** specifies that the file is to be accessed sequentially. **DIRECT** specifies that the file is to be accessed by record number. If **DIRECT** is specified, *iolist* must also contain a record length specifier. If *iolist* does not contain an access specifier, the value **SEQUENTIAL** is assumed. **KEYED** specifies that the file is accessed by a key−field value.**APPEND** specifies sequential access so that, after execution of an **OPEN** statement, the file is positioned after the last record.

ASSOCIATEVARIABLE=*asva*

specifies direct access only. After each I/O operation, *asvar* contains an integer variable giving the record number of the next sequential record number in the file. This parameter is ignored for all access modes other than direct access.

BLANK=*blnk* is a blank specifier. *blnk* is a character expression that, when all trailing blanks are removed, has the value **NULL** (the default) or **ZERO**. **NULL** ignores blank characters in numeric formatted input fields. **ZERO** specifies that all blanks other than leading blanks are to be treated as zeros. If *iolist* does not contain a blank specifier, the value **NULL** is assumed.

CARRIAGECONTROL=*type*

type is a character expression that determines carriage−control processing as shown in Table 8−5

Table 8−5 Carriage−Control Options

Value of *type*	Meaning
FORTRAN	Standard Fortran interpretation of the first character
LIST	Single spacing between lines
NONE	No implied carriage control

LIST is the default for formatted files, and **NONE** is the default for unformatted files. When the **-vms_cc** option (refer to Chapter 1 of the Fortran 77 Programmer's Guide) is specified, **FORTRAN** becomes the default for the standard output unit (unit 6).

DEFAULTFILE=*fname*

fname is either a character expression specifying a path name or an alternate prefix filename for the opened unit. When specified, the full filename of the opened unit is obtained by concatenating the string specified by *fname* with either the string in the **FILE** parameter (if specified) or with the unit number (when **FILE** is absent). *fname* can also be a numeric variable to which Hollerith data is assigned. A null character terminates the filename.

DISP[OSE]=*disposition*

disposition is a character expression that designates how the opened file is to be effect on the closed file.

Table 8–6 Disposition Options

Value for *disposition*	File status after CLOSE
KEEP	Retained (default)
SAVE	Same as KEEP
PRINT	Printed and retained.
PRINT/DELETE	Printed and deleted.
SUBMIT	Executed and retained
SUBMIT/DELETE	Executed and deleted

FORM=*fm* is a form specifier. *fm* is a character expression that, when all trailing blanks are removed has either the value **FORMATTED** or **UNFORMATTED**. The file opened with **FORMATTED** is connected for formatted I/O, and a file opened with **UNFORMATTED** is connected for unformatted I/O. The extensions **SYSTEM** and **BINARY** can also be used to specify the form of the file. A file opened with the **SYSTEM** specifier is unformatted and has no record marks. Data is written/read as specified by the I/O list with no record boundary, which is equivalent to opening a file with the **BINARY** specifier on the IRIS 3000 series. A file opened with **BINARY** allows unformatted binary records to be read and written using formatted **READ** and **WRITE** statements. This form is only needed if the **A** edit descriptor is used to dump out numeric binary data to the file.If *iolist* contains no form specifier, the default value is **FORMATTED** for sequential access files and **UNFORMATTED** for direct access files.

KEY=(*key1start*:*key1end*[: *type*] [, *key2start*:*key2end*[:*type*]]...)

defines the location and data type of one or more keys in an indexed record. The following rules apply to **KEY** parameters:

- At least one key (the primary key) must be specified when creating an indexed file.
- *type* is either **INTEGER** or **CHARACTER** (the default), defining the data type of the key.
- **INTEGER** keys must be specified with a length of 4.
- The maximum length of a key is 512 bytes.
- *key1start* and *key1end* are integers defining the starting and ending byte positions of the primary field, which is always required. *key2start* and *key2end* and subsequent specifications define the starting and ending positions of alternate fields, which are optional. There is no limit to the number of keys that can be specified.
- The sequence of the key fields determines the value in a key–of–reference specifier, **KEYID**, described in "Control Information List — cilist". **KEYID=0** specifies the field starting the *key1start* (primary) key; **KEYID=1** specifies the

field starting at *key2start*, and so forth.

- The **KEY** field must be specified only when an indexed file is created. The key specifications in the first **OPEN** remain in effect permanently for subsequent file openings. If **KEY** is specified when opening an existing file, the specifications must match those specified when the file was created.

MAXREC=*n* where *n* is a numeric expression defining the maximum number of records allowed in a direct–access file. If this parameter is omitted, no maximum limit exists.

RECL=*rl* is a record length specifier. *rl* is a positive integer expression specifying the length in characters or processor–dependent units for formatted and unformatted files, respectively. This specifier is required for direct–access files and keyed–access files; otherwise, it must be omitted.

READONLY specifies that the unit is to be opened for reading only. Other programs may open the file and have read–only access to it concurrently. If you do not specify this keyword, you can both read and write to the specified file.

RECORDSIZE=*rl*

has same effect as **RECL**.

RECORDTYPE=*rt*

when creating a file *rt* defines the type of records that the file is to contain; *rt* can be one of the following character expressions: **FIXED**, **VARIABLE**, or **STREAM_LF**. If **RECORDTYPE** is omitted, the default record type depends on the file type, as determined by the **ACCESS** and/or **FORM** parameters. The default types are shown in Table 8–7.

Table 8–7 Default Record Types

File Type	Record Type (Default)
Relative or indexed	FIXED
Direct–access sequential	FIXED
Formatted sequential access	STREAM_LF
Unformatted sequential access	VARIABLE

The following rules apply:

- If **RECORDTYPE** is specified, *rt* must be the appropriate default value shown in Table 8–7.
- When writing records to a fixed–length file, the record is padded with spaces (for formatted files) or with zeros (for unformatted files) when the output statement does not specify a full record.

SHARED ensures that the file is as up to date as possible by flushing each record as it is written.

STATUS=*sta* is a file status specifier. *sta* is a character expression that, ignoring trailing blanks, has one of the following values: **OLD** requires the **FILE=***fname* specifier, and it must exist. **NEW** requires the **FILE=***fname* specifier. The file is created by **OPEN**, and the file status is automatically turned to **OLD**. A file with the same name must not already exist. **SCRATCH** creates an unnamed file that is connected to the unit from **UNIT=** and will be deleted when that unit is closed by **CLOSE**. **DEFAULTFILE** can be used to specify a temporary directory to be used for opening the temporary file. Named files should not be used with **SCRATCH**. **UNKNOWN** meaning is processor dependent. See the *Fortran 77 Programmer's Guide* for more information. If the **STATUS** specifier is omitted, **UNKNOWN** is the default.

TYPE=*sta* is the same as **STATUS**.

Rules for Use

- Specifiers can be given in *iolist* in any order (for an exception, see the **UNIT** specifier on page 170).
- The unit specifier is required; all other specifiers are optional. The record–length specifier is required for connecting to a direct–access file.
- The unit specified must exist.
- An **OPEN** statement for a unit that is connected to an existing file is allowed. If the file specifier is not included, the file to be connected to the unit is the same as the file to which the unit is connected.
- A file to be connected to a unit that is not the same as the file currently connected to the unit has the same effect as a **CLOSE** statement without a file status specifier. The old file is closed, and the new one is opened.
- If the file to be connected is the same as the file to which the unit is currently connected, then all specifiers must have the same value as the current connection except the value of the **BLANK** specifier.
- See "Data Transfer Rules" for additional rules.

Examples

The following examples show the use of **OPEN** statements:

```
OPEN (1, STATUS='NEW')
OPEN (UNIT=1,STATUS='SCRATCH',ACCESS='DIRECT',RECL=64)
OPEN (1, FILE='MYSTUFF', STATUS='NEW',ERR=14,
 + ACCESS='DIRECT',RECL=1024)
OPEN (K,FILE='MAILLIST',ACCESS='INDEXED',FORM='FORMATTED',
+RECL=256,KEY=(1:20,21:30,31:35,200:256))
```

PRINT or TYPE

The **PRINT** (or **TYPE**) statement transfers data from the output list items to the file associated with the system output unit.

Syntax

PRINT *f* [,*iolist*]

where *f* is the format specifier and *iolist* is an optional output list specifying the data to be transferred as described in "Control Information List — cilist" and "Input/Output List — iolist".

TYPE is a synonym for **PRINT**.

Rules for Use

Use the **PRINT** statement to transfer formatted output to the system output unit. See "Data Transfer Rules" for additional rules.

Examples

The following examples show the use of **PRINT** and **TYPE** statements.

```
PRINT 10, (FORM (L), L=1,K+1)
PRINT *, X,Y,Z
TYPE *, ' VOLUME IS ',V,' RADIUS IS ',R
```

READ (Direct Access)

The direct−access **READ** statement transfers data from an external file to the items specified by the input list. Transfer occurs using the direct−access method. (See Chapter 7, "Input/Output Processing," for details about the direct access method.)

Syntax: Formatted

READ ([UNIT=]*unum*, REC=*rn*, *f* [,IOSTAT=*ios*] [,ERR=*s*]) [*iolist*]

Syntax: Unformatted

READ ([UNIT=]*unum*, REC=*rn*, [,IOSTAT=*rn*] [,ERR=*s*]) [*iolist*]

where

UNIT=*unum*	is a unit or internal file to be acted on.
f	is a format specifier.
REC=*rn*	is a direct−access mode. *rn* is the number of the record to be accessed.
IOSTAT=*rn*	is the name of variable in which I/O completion status is posted.
ERR=*s*	is the statement label to which control is transferred after an error.
iolist	specifies memory location where data is to be read.

See the "Control Information List — cilist" and "Input/Output List — iolist" for details on these parameters.

See "Control Information List — cilist", "Input/Output List — iolist", and Chapter 7, "Input/Output Processing," for more information on formatted and unformatted I/O.

READ (Indexed)

The indexed **READ** statement transfers data from an external indexed file to the items specified by the input list. Transfer occurs using the keyed access method. (See Chapter 7, "Input/Output Processing.")

Syntax: Formatted

READ[UNIT=]*unum*,*f*, KEY=*val*[,KEYID=*kn*][,IOSTAT=*rn*][,ERR=*s*])[*iolist*]

Syntax: Unformatted

READ ([UNIT=]*unum*,*key*[,*keyid*][,IOSTAT=*rn*] [,ERR=*s*]) [*iolist*]

where

UNIT=*unum*	is a unit or internal file to be acted upon.
f	is a format specifier.
KEY=*val*	is the value of the key field in the record to be accessed.
KEYID=*kn*	is the key reference specifier.
IOSTAT=*rn*	is the name of variable to which I/O completion status is posted.
ERR=*s*	is the statement label to which control is transferred after an error.
iolist	specifies memory location where data is read.

See "Control Information List — cilist" and "Input/Output List — iolist" for details on these parameters.

See "Data Transfer Rules" and Chapter 7, "Input/Output Processing," for more information on indexed I/O and the differences between formatted and unformatted I/O.

READ (Internal)

The internal **READ** statement transfers data from an internal file to internal storage.

Syntax: Formatted

READ ([UNIT=]*unum*,*f*[,IOSTAT=*rn*][,ERR=*s*][,END=*eof*]) [*iolist*]

Syntax: List–Directed

READ ([UNIT=]*unum*,* [,IOSTAT=*rn*][,ERR=*s*][,END=*eof*]) [*iolist*]

where

UNIT=*unum*	is a unit or internal file to be acted upon.
f	is a format specifier
*	is a list–directed input specifier.
IOSTAT=*rn*	is the name of variable in which I/O completion status is to be posted.

ERR=*s* is the statement label to which control is transferred after an error.

END=*eof* is the statement label to which control is transferred upon end−of−file.

iolist specifies memory location where data is to be read.

See "Control Information List — cilist" and "Input/Output List — iolist" for details on these parameters.

See "Data Transfer Rules" and Chapter 7, "Input/Output Processing," for more information on formatted and list−directed I/O. Chapter 7 also contains an example of I/O using internal files.

Note: The **DECODE** statement can also be used to control internal input. See "DECODE" for more information.

READ (Sequential)

The sequential **READ** statement transfers data from an external record to the items specified by the input list. Transfers occur using the sequential−access method or keyed−access method. (See Chapter 7, "Input/Output Processing.")

The four forms of the sequential **READ** statement are

- formatted
- list−directed
- unformatted
- namelist−directed

The following parameters apply to all four forms of the sequential **READ** statement:

[**UNIT=**]*unum*
is a unit or internal file to be acted upon.

f is a format specifier.

* is a list−directed input specifier.

NML=[*group−name*]
is a namelist specifier. If the keyword **NML** is omitted, *group−name* must immediately follow *unum*.

IOSTAT=*rn* is the name of the variable in which I/O completion status is posted.

ERR=*s* is the statement label to which control is transferred after an error.

END=*eof* is the statement label to which control is transferred on end of file.

iolist specifies memory location where data is read.

See "Control Information List — cilist" for details on these parameters.

Formatted READ (Sequential)

Syntax

```
READ ([UNIT=]unum,f[,IOSTAT=rn][,ERR=s] [,END=eof]) [iolist]
READ f[,iolist]
```

Method of Operation

A formatted **READ** statement transfers data from an external record to internal storage. It translates the data from character to binary format using the *f* specifier to edit the data.

List–Directed READ (Sequential)

Syntax

```
READ ([UNIT=]unum,*[,IOSTAT=rn][,ERR=s][,END=eof])[iolist]
READ f* [iolist]
```

Method of Operation

A list–directed**READ** statement transfers data from an external record to internal storage. It translates the data from character to binary format using the data types of the items in *iolist* to edit the data.

Rules for Use

- The external record can have one of the following values:
  - A constant with a data type of integer, real, logical, complex, or character. The rules given in Chapter 2, "Constants and Data Structures," define the acceptable formats for constants in the external record.
  - A null value, represented by a leading comma, two consecutive constants without intervening blanks, or a trailing comma.
  - A repetitive format *n*constant*, where *n* is a nonzero, unsigned integer constant indicating the number of occurrences of *constant*. *n** represents repetition of a null value.
- Hollerith, octal, and hexadecimal constants are not allowed.
- A value separator must delimit each item in the external record; a value separator can be one of the following:
  - one or more spaces or tabs
  - a comma, optionally surrounded by spaces or tabs
- A space, tab, comma, or slash appearing within a character constant are processed as part of the constant, not as delimiters.
- A slash delimits the end of the record and causes processing of an input statement to halt; the slash can be optionally surrounded by spaces and/or tabs. Any remaining items in *iolist* are unchanged after the **READ**.

- When the external record specified contains character constants, a slash must be specified to terminate record processing. If the external record ends with a blank, the first character of the next record processed follows immediately after the last character of the previous record.
- Each **READ** statement reads as many records as is required by the specifications in *iolist*. Any items in a record appearing after a slash are ignored.

Unformatted READ (Sequential)

Syntax

READ ([UNIT=]*unum*[,IOSTAT=*rn*][,ERR=*s*][,END=*eof*]) [*iolist*]

Method of Operation

An unformatted **READ** statement transfers data from an external record to internal storage. The **READ** operation performs no translation on read–in data. The data is read in directly to the items in *iolist*. The type of each data item in the input record must match that declared for the corresponding item in *iolist*.

When a sequential–unformatted **READ** is performed on a direct–access file, the next record in the direct–access file is assumed.

Rules for Use

- There must be at least as many items in the unformatted record as there are in *iolist*. Additional items in the record are ignored, and a subsequent **READ** accesses the next record in the file.
- The type of each data item in the i3nput record must match the corresponding data item in *iolist*.

Namelist–Directed READ (Sequential)

Syntax

READ (*unum*,NML=*group–name*[,IOSTAT=*rn*][,ERR=*s*][,END=*eof*]) [*iolist*]
READ *name*

Method of Operation

A namelist–directed **READ** statement locates data in a file using the group name in a **NAMELIST** statement (see Chapter 4, "Specification Statements.") It uses the data types of the items in the corresponding **NAMELIST** statement and the forms of the data to edit the data.

Figure 8–1 illustrates rules for namelist input data and shows its format.

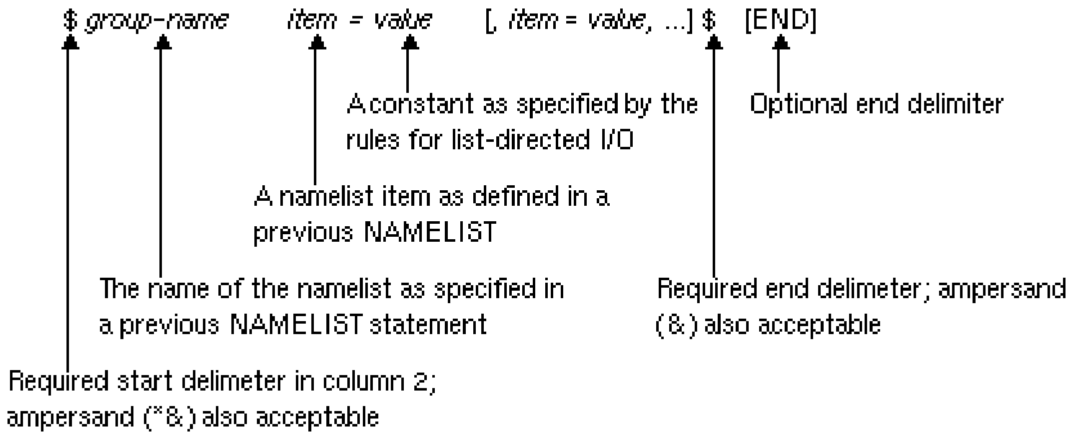

Figure 8–1 Namelist Input Data Rules

Rules for Use

- Both *group–name* and *item* must be contained within a single record.
- Spaces and/or tabs are not allowed within *group–name* or *item*. However, *item* can contain spaces or tabs within the parentheses of a subscript or substring specifier.
- The *value* item can be any of the values given under the first rule in the previous section, "List–Directed READ (Sequential)".
- A *value* separator must delimit each item in a list of constants. See the third and fourth rules in "List–Directed READ (Sequential)".
- A separator must delimit each list of value assignments. See the third rule in "List–Directed READ (Sequential)". Any number of spaces or tabs can precede the equal sign.
- When *value* contains character constants, a dollar sign ($) or ampersand (&) must be specified to terminate processing of the namelist input. If the namelist input ends with a blank, the first character of the next record processed follows immediately after the last character of the previous record.
- Entering a question mark (?) after a namelist–directed **READ** statement is executed causes the *group–name* and current values of the namelist items for that group to be displayed.
- You can assign input values in any order in the format *item=value*. Multiple–line assignment statements are allowed. Each new line must begin on or after column 2; column 1 is assumed to contain a carriage–control character. Any other character in column 1 is ignored.
- You can assign input values for the following data types: integer, real, logical, complex, and character. Refer to Table 5–2 in Chapter 5 for the conversion rules when the data type of the namelist item and the assigned constant value do not match.
- Numeric–to–character and character–to–numeric conversions are not allowed.

- Constant values must be given for assigned values, array subscripts, and substring specifiers. Symbolic constants defined by a **PARAMETER** statement are not allowed.

Example

In the following example, the name of a file is read from the standard input into **filename**, the file is opened, and the first record is read. A branch is taken to statement 45 (not shown) when end of file is encountered.

```
   read (*,10) filename
10 format (a)
   open (2,file=filename)
   read (2, 20, end=45) word
20 format (A50)
```

See "Data Transfer Rules" and Chapter 7, "Input/Output Processing," for more information on formatted, list−directed unformatted, and namelist−directed I/O.

REWIND

The **REWIND** statement positions a file at its initial point. It can be used with both unformatted and formatted data files.

Syntax

REWIND *u*
REWIND (*alist*)

where

u	is an external unit identifier.
alist	is a list of the following specifiers:
[**UNIT =**] *u*	is a required unit specifier. *u* must be an integer expression that identifies the number of an external unit. If the keyword **UNIT=** is omitted, then *u* must be the first specifier in *alist*.
IOSTAT = *ios*	is an I/O status specifier that specifies the variable to be defined with a status value by the **REWIND** statement. A zero value for *ios* denotes a no error condition, while a positive integer value denotes an error condition.
ERR = *s*	is an error specifier that identifies a statement number to which control is transferred when an error condition occurs during the execution of the **REWIND** statement.

Method of Operation

The unit specifier is required and must appear exactly once. The other specifiers are optional and can appear at most once each in the *alist*. Specifiers can appear in any order (refer to"Unit Specifier— UNIT" for exceptions). The **REWIND** statement positions the specified file at its initial point. If the file is already at its initial point, the **REWIND** statement has no effect. It is legal to specify a **REWIND** statement for a file that is connected but does not exist, but the statement has no effect.

Examples

The following statements show examples of the **REWIND** statement.

```
REWIND 8
REWIND (UNIT=NFILE,ERR=555)
```

REWRITE

The **REWRITE** statement transfers data to an external indexed file from the items specified by the output list. The record transferred is the last record accessed from the same file using an indexed **READ** statement.

Syntax: Formatted

REWRITE ([UNIT=]*unum*,*f*[,IOSTAT=*rn*][,ERR=*s*]) [*iolist*]

Syntax: Unformatted

REWRITE ([UNIT=]*unum*[,IOSTAT=*rn*][,ERR=*s*]) [*iolist*]

where

[**UNIT=**]*unum*	is the unit or internal file to be acted on.
f	is a format specifier.
IOSTAT=*rn*	is the name of a variable in which I/O completion status is posted.
ERR=*s*	is a statement label to which control is transferred after an error.

See "Control Information List — cilist" and "Input/Output List — iolist" for details on these parameters.

Rules for Use

The **REWRITE** statement is supported for both formatted and unformatted indexed files. The statement provides a means for changing existing records in the file.

See "Data Transfer Rules" for additional rules.

Example

```
REWRITE (10), A,B,C
```

The previous statement rewrites the last record accessed to the indexed file connected to logical unit 10.

UNLOCK

The **UNLOCK** statement makes the last record read from an indexed file available for access by other users.

Syntax

```
UNLOCK [UNIT=]unum
UNLOCK ([UNIT=]unum[,IOSTAT=rn][,ERR=s]
```

where

UNIT=*unum* is a unit or internal file to be acted on.

IOSTAT=*rn* is the name of variable in which I/O completion status is posted.

ERR=*s* is the statement label to which control is transferred after an error.

See "Control Information List — cilist" for details on each of these parameters.

Method of Operation

After a record is read from an indexed file, it cannot be accessed by other users until an **UNLOCK** statement is executed, the record is rewritten, or a new record is read.

Example

The following statement unlocks the last record read in from the file connected to logical unit 10.

```
UNLOCK (10)
```

WRITE (Direct Access)

The direct–access **WRITE** statement transfers data from internal storage to an external indexed file using the direct–access method.

Syntax: Formatted

```
WRITE ([UNIT=]unum,REC=rn,f[,IOSTAT=rn][,ERR=s]) [iolist]
```

Syntax: Unformatted

```
WRITE ([UNIT=]unum,REC=rn[,IOSTAT=ios][,ERR=s]) [iolist]
```

where

[**UNIT=**]*unum* is a unit or internal file to be acted upon.

REC=*rn* is a direct–access mode. *rn* is the number of the record to be accessed.

f is a format specifier.

IOSTAT=*rn* is the name of variable in which I/O completion status is posted.

ERR=*s* is the statement label to which control is transferred after an error.

iolist specifies memory location from which data is written.

See "Control Information List — cilist" and "Input/Output List — iolist" for details on these parameters.

See "Data Transfer Rules" and Chapter 7, "Input/Output Processing," for more information on formatted and unformatted I/O.

Rules for Use

Execution of a **WRITE** statement for a file that does not exist creates the file.

WRITE (Indexed)

The indexed **WRITE** statement transfers data from internal storage to external records using the keyed–access method.

Syntax: Formatted

`WRITE ([UNIT=]`*unum*`,`*f*`[,IOSTAT=`*rn*`][,ERR=`*s*`]) [`*iolist*`]`

Syntax: Unformatted

`WRITE ([UNIT=]`*unum*`[,IOSTAT=`*rn*`][,ERR=`*s*`]) [`*iolist*`]`

where

[**UNIT=**]*unum*	is a unit or internal file to be acted on.
f	is a format specifier.
*	is the list–directed output specifier.
IOSTAT=*rn*	is the name of a variable in which I/O completion status is posted.
ERR=s	is a statement label to which control is transferred after an error.
iolist	specifies a memory location from which data is written.

See "Control Information List — cilist" and "Input/Output List — iolist" for details on these parameters.

See "Data Transfer Rules" and Chapter 7, "Input/Output Processing," for more information on formatted and unformatted I/O.

Rules for Use

Execution of a **WRITE** statement for a file that does not exist creates the file.

WRITE (Internal)

The internal **WRITE** statement transfers data to an external file or an internal file from the items specified by the output list.

Syntax: Formatted

`WRITE ([UNIT=]`*unum*`,`*f*`[,IOSTAT=`*ios*`][,ERR=`*s*`])[`*iolist*`]`

Syntax: List–directed

```
WRITE ([UNIT=]unum, *[,IOSTAT=rn][,ERR=s])[iolist]
```

where

[**UNIT=**]*unum*	is a unit or internal file to be acted on.
f	is a format specifier.
*	is the list–directed output specifier.
IOSTAT=*rn*	is the name of a variable in which I/O completion status is posted.
ERR=*s*	is the statement label to which control is transferred after an error.
iolist	specifies a memory location from which data is written.

See "Control Information List — cilist" and"Input/Output List — iolist" for details on these parameters.

See "Data Transfer Rules" and Chapter 7, "Input/Output Processing," for more information on formatted and list–directed I/O. Chapter 7 also contains an example of I/O using internal files.

Rules for Use

Execution of an internal **WRITE** statement for a file that does not exist creates the file.

Note: The **ENCODE** statement can also be used to control internal output. See the **ENCODE** statement description on page 161 for more information.

WRITE (Sequential)

The sequential **WRITE** statement transfers data to an external file or an internal file from the items specified by the output list.

The four types of sequential **WRITE** statements are

- formatted
- unformatted
- list–directed
- namelist–directed

Each of these statements is discussed in the following sections.

Execution of a **WRITE** statement for a file that does not exist creates the file.

Parameter Explanations

UNIT=*unum* is a unit or internal file to be acted on.

NML= *group–name*
is a namelist specifier.

f	is a format specifier.
*	is the list–directed output specifier.
REC=*rn*	is a direct–access mode.*rn* is the number of the record to be accessed.
IOSTAT=*rn*	is the name of a variable in which I/O completion status is posted.
ERR=*s*	is a statement label to which control is transferred after an error.
iolist	specifies a memory location from which data is written.

See "Control Information List — cilist" and "Input/Output List — iolist" for details on these parameters.

See "Data Transfer Rules" and Chapter 7, "Input/Output Processing," for more information on formatted, list–directed, and unformatted I/O.

Formatted WRITE (Sequential)

```
WRITE ([UNIT=]unum,f[,IOSTAT=rn][,ERR=s]) [iolist]
```

Method of Operation

A formatted **WRITE** statement transfers data from internal storage to an external record using sequential–access mode. The**WRITE** operation translates the data from binary to character format using the*f* specifier to edit the data.

Unformatted WRITE (Sequential)

```
WRITE ([UNIT=]unum[,IOSTAT=rn][,ERR=s]) [iolist]
```

Method of Operation

An unformatted **WRITE** statement performs no translation on read–in data. The data is read in directly to the items in *iolist*. The type of each data item in the input record must match that declared for the corresponding item in *iolist*.

When sequential–formatted**WRITE** is performed on a direct–access file, the next record in the file is assumed and the record is zero–padded to the end as if it were a direct, unformatted**WRITE**.

List–Directed WRITE

```
WRITE ([UNIT=]unum,*[,IOSTAT=rn][,ERR=s]) [iolist]
```

Method of Operation

A list–directed**WRITE** statement transfers data from internal storage to an external record using sequential–access mode. The**WRITE** operation translates the data from binary to character format using the data types of the items in *iolist* to edit the data.

Rules

- The item to be transferred to an external record can be a constant with a data type of integer, real, logical, complex, or character.
- The rules given in Chapter 2, "Constants and Data Structures,"define the acceptable formats for constants in the external record, except character constant. A character constant does not require delimiting apostrophes; an apostrophe within a character string is represented by one apostrophes instead of two.

Table 8−8shows the data types and the defaults of their output format.

Table 8−8 Default Formats of List−Directed Output

Data Type	Format Specification of Default Output
BYTE	L2
LOGICAL*1	I5
LOGICAL*2	L2
LOGICAL*4	L2
INTEGER*1	I5
INTEGER*2	I7
INTEGER*4	i12
REAL*4	1pg15.7e2
REAL*8	1pg24.16e2
COMPLEX	'(',1pg15.7e2,',',1pg15.7e2,')'
COMPLEX*16	'(',1pg24.16e2,',',1pg24.16e2,')'
CHARACTER**n*	A*n*, where *n* is the length of the character expression

- List−directed character output data cannot be read as list−directed input because of the use of apostrophes described above.
- A list−directed output statement can write one or more records. Position one of each record must contain a space (blank), which Fortran uses for a carriage−control character. Each value must be contained within a single record with the following exceptions:
  - A character constant longer than a record can be extended to a second record.
  - A complex constant can be split onto a second record after the comma.
- The output of a complex value contains no embedded spaces.
- Octal values, null values, slash separators, or the output of a constant or null value in the repetitive format *n*constant* or *n*z* cannot be generated by a list−directed output statement.

Namelist−Directed WRITE

Syntax

WRITE([UNIT=]*unum*,NML=*group−name*[,IOSTAT=*rn*][,ERR=*s*][,END=*eof*])

Method of Operation

A namelist−directed **WRITE** statement transfers data from internal storage to external records. It translates the data from internal to external format using the data type of the items in the corresponding **NAMELIST** statement (see Chapter 4, "Specification Statements.") A namelist−directed **READ** or **ACCEPT** statement can read the output of a namelist−directed **WRITE** statement.

Rules for Use

Namelist items are written in the order that referenced **NAMELIST** defines them.

Examples for All Forms of Sequential WRITE

The following statement writes the prompt **enter a filename** to standard output:

```
    write (*,105)
105 format (1x,'enter a filename')
```

The following statement opens the file **%%temp** and writes the record **pair** to the file.

```
     open (unit=10, status='unknown',file="%%temp")
     write (10,1910) pair
1910 format (A)
```

Control Information List — cilist

This section describes the components of the control information list (*cilist*) and the I/O list (*iolist*), which can be specified as elements of the I/O statements described in this chapter.

Table 8−9 summarizes the items that can be specified in a cilist. Each *cilist* specifier shown in the table can appear no more than once in a *cilist*. Note that the keywords **UNIT=** and **FMT=** are optional. Normally, the *cilist* items may be written in any order, but if **UNIT=** or **FMT=** is omitted, the following restrictions apply:

- The keyword **UNIT=** can be omitted if and only if the unit specifier is the first item on the list.
- The keyword **FMT=** can be omitted if and only if the format specifier is the second item in the *cilist* and the first item is a unit specifier in which the keyword **UNIT=** has been omitted.

  A format specifier denotes a formatted I/O operation; default is an unformatted I/O operation. If a record specifier is present, then direct access I/O is denoted; default is sequential access.

Table 8−9 Control Information List Specifiers

Specifier	Purpose
[UNIT=]*u*	Unit or internal file to be acted on.
[NML= *group−name*]	Identifies the *group−name* of a list of items for namelist−directed I/O.
[FMT=]*f*	Formatted or unformatted I/O operations. If formatted, contains format specifiers for data to be read or written.
REC= *rn*	Number of a record accessed in direct−access mode.
KEY [*c*] =*val*	Value of the key field in a record accessed in indexed access mode, where c can be the optional match condition EQ, GT, or GE.

KEYID= *kn*	Key−reference specifier, specifying either the primary key or one of the alternate keys in a record referenced in indexed−access mode.
IOSTAT= *ios*	Name of a variable in which I/O completion status is returned.
ERR= *s*	Label of a statement to which control is transferred if an error occurs.
END= *s*	Label of a statement to which control is transferred if an end−of−file condition (READ only) occurs.

Unit Specifier — UNIT

The form of a unit specifier is

`[UNIT=]`*u*

where *u* is a unit identifier specified as follows:

- A nonnegative integer or noninteger expression specifying the unit.

  A noninteger expression is converted to integer, and the fractional portion, if present, is discarded before use.

- An asterisk specifying a unit that is connected for formatted sequential access (external file identifier only). This denotes the system input unit in a **READ** statement or the system output unit in a **WRITE** statement.

- An identifier that is the name of a character variable, character array, character array element, or substring (internal file identifier only).

An external unit identifier can have the form described in the first or second rule above, except that it cannot be an asterisk in an auxiliary output statement.

An internal file identifier must be specified in the third rule above.

The syntax shows that you can omit the **UNIT=** keyword. If **UNIT=** is omitted, the unit identifier must be first in a control information list. For example, two equivalent **READ** statements are

```
READ(UNIT=5)
READ(5)
```

Format Specifier — FMT

The syntax of a format specifier is

`[FMT=]`*f*

where

f	is a format identifier. As shown in the syntax, the keyword **FMT=** can be omitted from the format identifier. If so, the format identifier must be second in a control information list, and the **UNIT=** keyword must also have been omitted.

The legal kinds of format identifiers are

- the statement label of a **FORMAT** statement (the **FORMAT** statement and the format identifier must be in the same program unit)

- an integer variable name assigned to the statement label of a **FORMAT** statement (the **FORMAT** statement and the format identifier must be in the same program unit)
- a character expression (provided it does not contain the concatenation of a dummy argument that has its length specified by an asterisk)
- the name of a character array
- an asterisk that is used to indicate list−directed formatting

Namelist Specifier — NML

The namelist specifier indicates namelist−directed I/O within the **READ** or **WRITE** statement where **NML** is specified. It has the format

`[NML=]`*group−name*

where *group−name* identifies the list in a previously defined **NAMELIST** statement (see Chapter 4, "Specification Statements.")

NML can be omitted when preceded by a unit specifier (unum) without the optional **UNIT** keyword.

Record Specifier — REC

The form of a record specifier is

`REC=`*rn*

where *rn* is an expression that evaluates the record number of the record to be accessed in a direct−access I/O operation. Record numbers must be integers greater than zero.

Key−Field−Value Specifier — KEY

The indexed−access method uses the key−field−value specified in **READ**, **REWRITE**, or other I/O statement. A key field in the record is used as criteria in selecting a record from an indexed file. The key fields for the records in an indexed file are established by the **KEY** specifier used in the **OPEN** statement that created the file.

The key−field−value specifier has the forms shown in Table 8−10

Table 8−10 Forms of the Key−Field−Value Specifier

Specifier	Basis for Record Selection
KEY=*kval*	Key−field value *kval*
KEYEQ=*kval*	Key−field value *kval* and the key field are equal
KEYGT=*kval*	Key−field value is greater than the key field
KEYGE=*kval*	Key−field value is greater than or equal to the key field

The following rules apply to *kval*:

- *kval* can be a character or integer expression; if an integer expression, it cannot contain any real or complex values. If the indexed file is formatted, *kval* should always be a character expression.

- The character expression can be an ordinary character string or an array name of type **LOGICAL*1** or **BYTE** containing Hollerith data.
- The character or integer type specified for *kval* must match the type specified for the key field in the record.

Key−of−Reference Specifier — KEYID

The key−of−reference specifier designates, in a **READ**, **REWRITE**, or other I/O statement, the key field in a record to which the key−field−value specifier applies.

The specifier has the following format:

`KEYID=`*n*

where *n* is a number from 0 to the maximum number of keys defined for the records in the indexed file; 0 specifies the primary key, 1 specifies the first alternate key, 2 specifies the second alternate key, and so on. The **KEY** parameter of the **OPEN** statement that created the files creates and establishes the ordering of the primary and alternate keys.

If **KEYID** is not specified, the previous **KEYID** specification in an I/O statement to the same I/O unit is used. The default for **KEYID** is zero (0) if it is not specified for the first I/O statement.

Input/Output Status Specifier — ios

An I/O status specifier has the form

`IOSTAT=`*ios*

where *ios* is a status variable indicating an integer variable or an integer array element. Execution of an I/O statement containing this specifier causes *ios* to become defined with one of the following values:

- Zero if neither an error condition nor an end−of−file condition is encountered by the processor, indicating a successful operation
- Positive integer if an error condition occurred
- Negative integer if an end−of−file condition is encountered without an error condition

  For details about **IOSTAT**, refer to the *perror*(3F) and *intro*(2) manual pages.

Error Specifier — ERR

An error specifier has the following form

`ERR=`*s*

where *s* is an error return label of an executable statement that appears in the same program unit as the error specifier.

If an error condition occurs during execution of an I/O statement with an error specifier, execution of the statement is terminated and the file position becomes indeterminate. If the statement contains an I/O status specifier, the status variable *ios* becomes defined with a processor−dependent positive integer. Execution then continues at the statement labeled *s*.

End−of−File Specifier — END

The form of an end−of−file specifier is

```
END=s
```

where *s* is an end−of−file return label of an executable statement that appears in the same program unit as the end−of−file specifier. An end−of−file specifier may only be used on the *cilist* of a **READ** statement.

If an end−of−file condition is encountered during the execution of a **READ** statement containing an end−of−file specifier and no error occurs, execution of the **READ** statement terminates. If the **READ** statement contains an I/O status specifier, the I/O status variable *ios* becomes defined with a processor−dependent negative integer. Execution then continues at the statement labeled *s*.

Input/Output List — iolist

This section describes the components of I/O list (*iolist*), which can be specified as elements of the I/O statements described in this chapter.

An input/output list specifies the memory locations of the data to be transferred by the I/O statements **READ**, **WRITE**, and **PRINT**.

If an array name is given as an I/O list item, the elements in the array are treated as though each element were explicitly specified in the I/O list in storage order. Note that the name of an assumed−size dummy array (that is, an array declared with an * for an upper bound) must not appear as an I/O list item.

Input List

An input list item can be one of the following:

- Variable name.
- Array element name.
- Substring name.
- Array name.
- Implied **DO** list containing any of the above and other implied−**DO** lists.
- An aggregate reference (a structured data item as defined by a **RECORD** and **STRUCTURE** statement). An aggregate reference can be used only in unformatted input statements. When an aggregate name appears in an *iolist*, only one record is read regardless of how many aggregates or other list items are present.

Examples of input lists are

```
READ(5,3000,END=2000)X,Y(J,K+3),C(2:4)
READ(JFILE,REC=KNUM,ERR=1200)M,SLIST(M,3),cilist
```

Output List

An output list item can be one of the following:

- Variable name.
- Array element name.
- Substring name.
- Array name.
- Any expression, except a character expression involving concatenation of an operand with a length specification of asterisk (*), unless the operand is the symbolic name of a constant.
- An implied **DO** list containing any of the above and other implied **DO** lists.
- An aggregate reference (a structured data item as defined by a **RECORD** and **STRUCTURE** statement). An aggregate reference can be used only in unformatted output statements. When an aggregate name appears in an *iolist*, only one record is written regardless of how many aggregates or other list items are present.

Note that a constant, an expression involving operators or function references, or an expression enclosed in parentheses may appear in an output list but not in an input list.

An example of an output list is

```
WRITE(5,200,ERR=10)'ANSWER IS',N,SQRT(X)+1.23
```

Implied−DO Lists

An implied−**DO** list is a specification that follows the I/O list (*iolist*) in an I/O statement. The list permits the iteration of the statement as though it were contained within a **DO** loop. An implied−**DO** list has the form:

`(`*iolist*`,i=`*e1*`,`*e2*`[,`*e3*`])`

where

iolist	is one or more valid names of the data to be acted on.
i	is an iteration count.
e1, *e2*, and *e3*	are control parameters. See the description of the **DO** statement in Chapter 6, "Control Statements," for a description of *i*, *e1*, *e2*, and *e3*.

The control variable i must not appear as an input list item in *iolist*. The list items in *iolist* are specified once for each iteration of the implied **DO** list with the appropriate substitution of values for each occurrence of the control variable i. When an I/O error occurs within the implied **DO** loop, the value of the control variable i is undefined.

Example

The following statements write **Hello World** to standard output 100 times:

```
write (*,111) ('Hello World',i=1,100)
111 format (1x,A)
end
```

Data Transfer Rules

Data are transferred between records and items specified by the I/O list. The list items are processed in the order in which they appear in the list.

The following restrictions apply to data transfer operations:

- An input list item must not contain any portion of the established format specification.
- If an internal file has been specified, an I/O list item must not be in the file or associated with the file.
- Each output list item must be defined before the transfer of that item.
- All values needed to determine which entities are specified by an I/O list item are determined at the beginning of the processing of that item.

The following sections discuss the rules specific to unformatted and formatted I/O.

Unformatted Input/Output

The execution of an unformatted I/O statement transfers data without editing between the current record and the items specified in the I/O list. Exactly one record is either read or written.

For an unformatted input statement, the record must contain at least as many values as the number of values required by the input list. The data types of the values in the record must agree with the types of the corresponding items in the input list. Character data from an input record must have the same length attribute as the corresponding item in the input list.

The following conventions apply to the execution of an unformatted output statement:

- For direct access, the output list must not specify more values than can fit into a record. If the values specified by the output list do not fill the record, the remainder of the record is filled with zeros.
- For sequential access, the output list defines the size of the output record.

Fortran 77 allows unformatted data transfer only for external files and prohibits it for files connected for formatted I/O.

Formatted Input/Output

The execution of a formatted I/O statement transfers data with editing between the items specified by the I/O list and the file. The current record and possibly additional records are read or written.

Each execution of a **READ** statement causes at least one record to be read, and the input list determines the amount of data to be transferred from the record. The position and form of that data are established by the corresponding format specification.

In a formatted output operation, each execution of the **WRITE** or **PRINT** statement causes at least one record to be written. The amount of data written to the specified unit is determined both by the output list and the format specification.

When a repeatable edit descriptor in a format specification is encountered, a check is made for the existence of a corresponding item in the I/O list. If there is such an item, it transmits appropriately edited information between the item and the record, and then format control proceeds. If there is no corresponding item, format control terminates. Chapter 9, "Format Specification," explains formatted I/O in detail.