Section 9: Input/output statements

Input statements provide the means of transferring data from external media to internal storage or from an internal file to internal storage. This process is called **reading**. **Output statements** provide the means of transferring data from internal storage to external media or from internal storage to an internal file. This process is called **writing**. Some input/output statements specify that editing of the data is to be performed.

In addition to the statements that transfer data, there are auxiliary input/output statements to manipulate the external medium, or to describe or inquire about the properties of the connection to the external medium.

The input/output statements are the OPEN, CLOSE, READ, WRITE, PRINT, BACKSPACE, ENDFILE, REWIND, and INQUIRE statements.

The READ statement is a **data transfer input statement**. The WRITE statement and the PRINT statement are **data transfer output statements**. The OPEN statement and the CLOSE statement are **file connection statements**. The INQUIRE statement is a **file inquiry statement**. The BACKSPACE, ENDFILE, and REWIND statements are **file positioning statements**.

9.1 Records

A **record** is a sequence of values or a sequence of characters. For example, a line on a terminal is usually considered to be a record. However, a record does not necessarily correspond to a physical entity. There are three kinds of records:

(1) Formatted
(2) Unformatted
(3) Endfile

NOTE 9.1

What is called a "record" in Fortran is commonly called a "logical record". There is no concept in Fortran of a "physical record."

9.1.1 Formatted record

A **formatted record** consists of a sequence of characters that are capable of representation in the processor; however, a processor may prohibit some control characters (3.1) from appearing in a formatted record. The length of a formatted record is measured in characters and depends primarily on the number of characters put into the record when it is written. However, it may depend on the processor and the external medium. The length may be zero. Formatted records may be read or written only by formatted input/output statements.

Formatted records may be prepared by means other than Fortran; for example, by some manual input device.

9.1.2 Unformatted record

An **unformatted record** consists of a sequence of values in a processor-dependent form and may contain data of any type or may contain no data. The length of an unformatted record is measured in processor-dependent units and depends on the output list (9.4.2) used when it is written, as well as on the processor and the external medium. The length may be zero. Unformatted records may be read or written only by unformatted input/output statements.

9.1.3 Endfile record

An **endfile record** is written explicitly by the ENDFILE statement; the file shall be connected for sequential access. An endfile record is written implicitly to a file connected for sequential access when the most recent data transfer statement referring to the file is a data transfer output statement, no intervening file positioning statement referring to the file has been executed, and

1. A REWIND or BACKSPACE statement references the unit to which the file is connected or
2. The unit (file) is closed, either explicitly by a CLOSE statement, implicitly by a program termination not caused by an error condition, or implicitly by another OPEN statement for the same unit.

An endfile record may occur only as the last record of a file. An endfile record does not have a length property.

NOTE 9.2

> An endfile record does not necessarily have any physical embodiment. The processor may use a record count or other means to register the position of the file at the time an ENDFILE statement is executed, so that it can take appropriate action when that position is reached again during a read operation. The endfile record, however it is implemented, is considered to exist for the BACKSPACE statement (9.5.1).

9.2 Files

A **file** is a sequence of records.

There are two kinds of files:

1. External
2. Internal

NOTE 9.3

> For more explanatory information on files, see C.6.1.

9.2.1 External files

An **external file** is any file that exists in a medium external to the program.

At any given time, there is a processor-dependent set of allowed **access methods**, a processor-dependent set of allowed **forms**, a processor-dependent set of allowed **actions**, and a processor-dependent set of allowed **record lengths** for a file.

NOTE 9.4

> An example of a restriction on input/output statements (9.8) is that an input statement shall not specify that data are to be read from a printer.

A file may have a name; a file that has a name is called a **named file**. The name of a named file is a character string. The set of allowable names for a file is processor dependent.

An external file that is connected to a unit has a **position** property (9.2.1.3).

9.2.1.1 File existence

At any given time, there is a processor-dependent set of external files that are said to **exist** for a program. A file may be known to the processor, yet not exist for a program at a particular time.

NOTE 9.5

> Security reasons may prevent a file from existing for a program. A newly created file may exist but contain no records.

To create a file means to cause a file to exist that did not exist previously. To delete a file means to terminate the existence of the file.

All input/output statements may refer to files that exist. An INQUIRE, OPEN, CLOSE, WRITE, PRINT, REWIND, or ENDFILE statement also may refer to a file that does not exist. Execution of a WRITE, PRINT, or ENDFILE statement referring to a preconnected file that does not exist creates the file.

9.2.1.2 File access

There are two methods of accessing the records of an external file, sequential and direct. Some files may have more than one allowed access method; other files may be restricted to one access method.

NOTE 9.6

For example, a processor may allow only sequential access to a file on magnetic tape. Thus, the set of allowed access methods depends on the file and the processor.

The method of accessing the file is determined when the file is connected to a unit (9.3.2) or when the file is created if the file is preconnected (9.3.3).

9.2.1.2.1 Sequential access

When connected for **sequential access**, an external file has the following properties:

(1) The order of the records is the order in which they were written if the direct access method is not a member of the set of allowed access methods for the file. If the direct access method is also a member of the set of allowed access methods for the file, the order of the records is the same as that specified for direct access. In this case, the first record accessible by sequential access is the record whose record number is 1 for direct access. The second record accessible by sequential access is the record whose record number is 2 for direct access, etc. A record that has not been written since the file was created shall not be read.

(2) The records of the file are either all formatted or all unformatted, except that the last record of the file may be an endfile record. Unless the previous reference to the file was a data transfer output statement or a file positioning statement, the last record, if any, of the file shall be an endfile record.

(3) The records of the file shall not be read or written by direct access input/output statements.

9.2.1.2.2 Direct access

When connected for **direct access**, an external file has the following properties:

(1) Each record of the file is uniquely identified by a positive integer called the **record number**. The record number of a record is specified when the record is written. Once established, the record number of a record can never be changed. The order of the records is the order of their record numbers.

NOTE 9.7

A record may not be deleted; however, a record may be rewritten.

(2) The records of the file are either all formatted or all unformatted. If the sequential access method is also a member of the set of allowed access methods for the file, its endfile record, if any, is not considered to be part of the file while it is connected for direct access. If the sequential access method is not a member of the set of allowed access methods for the file, the file shall not contain an endfile record.

(3) Reading and writing records is accomplished only by direct access input/output statements.

(4) All records of the file have the same length.

(5) Records need not be read or written in the order of their record numbers. Any record may be written into the file while it is connected to a unit. For example, it is permissible to write record 3, even though records 1 and 2 have not been written. Any record may be read from the file while it is connected to a unit, provided that the record has been written since the file was created.

(6) The records of the file shall not be read or written using list-directed formatting (10.8), namelist formatting (10.9), or a nonadvancing input/output statement (9.2.1.3.1).

9.2.1.3 File position

Execution of certain input/output statements affects the position of an external file. Certain circumstances can cause the position of a file to become indeterminate.

The **initial point** of a file is the position just before the first record. The **terminal point** is the position just after the last record. If there are no records in the file, the initial point and the terminal point are the same position.

If a file is positioned within a record, that record is the **current record**; otherwise, there is no current record.

Let n be the number of records in the file. If $1 < i \le n$ and a file is positioned within the ith record or between the $(i-1)$th record and the ith record, the $(i-1)$th record is the **preceding record**. If $n \ge 1$ and the file is positioned at its terminal point, the preceding record is the nth and last record. If $n = 0$ or if a file is positioned at its initial point or within the first record, there is no preceding record.

If $1 \le i < n$ and a file is positioned within the ith record or between the ith and $(i+1)$th record, the $(i+1)$th record is the **next record**. If $n \ge 1$ and the file is positioned at its initial point, the first record is the next record. If $n = 0$ or if a file is positioned at its terminal point or within the nth (last) record, there is no next record.

9.2.1.3.1 Advancing and nonadvancing input/output

An **advancing input/output statement** always positions the file after the last record read or written, unless there is an error condition.

A **nonadvancing input/output statement** may position the file at a character position within the current record, or a subsequent record (10.6.2). Using nonadvancing input/output, it is possible to read or write a record of the file by a sequence of input/output statements, each accessing a portion of the record. It is also possible to read variable-length records and be notified of their lengths.

9.2.1.3.2 File position prior to data transfer

The positioning of the file prior to data transfer depends on the method of access: sequential or direct.

For sequential access on input, if there is a current record, the file position is not changed. Otherwise, the file is positioned at the beginning of the next record and this record becomes the current record. Input shall not occur if there is no next record or if there is a current record and the last data transfer statement accessing the file performed output.

If the file contains an endfile record, the file shall not be positioned after the endfile record prior to data transfer. However, a REWIND or BACKSPACE statement may be used to reposition the file.

For sequential access on output, if there is a current record, the file position is not changed and the current record becomes the last record of the file. Otherwise, a new record is created as the next record of the file; this new record becomes the last and current record of the file and the file is positioned at the beginning of this record.

For direct access, the file is positioned at the beginning of the record specified by the record specifier. This record becomes the current record.

9.2.1.3.3 File position after data transfer

If an error condition (9.4.3) occurred, the position of the file is indeterminate. If no error condition occurred, but an end-of-file condition (9.4.3) occurred as a result of reading an endfile record, the file is positioned after the endfile record.

For nonadvancing input, if no error condition or end-of-file condition occurred, but an end-of-record condition (9.4.3) occurred, the file is positioned after the record just read. If no error condition, end-of-file condition, or end-of-record condition occurred in a nonadvancing input statement, the file position is not changed. If no error condition occurred in a nonadvancing output statement, the file position is not changed. In all other cases, the file is positioned after the record just read or written and that record becomes the preceding record.

9.2.2 Internal files

Internal files provide a means of transferring and converting data from internal storage to internal storage.

9.2.2.1 Internal file properties

An internal file has the following properties:

(1) The file is a variable of default character type that is not an array section with a vector subscript.

(2) A record of an internal file is a scalar character variable.

(3) If the file is a scalar character variable, it consists of a single record whose length is the same as the length of the scalar character variable. If the file is a character array, it is treated as a sequence of character array elements. Each array element, if any, is a record of the file. The ordering of the records of the file is the same as the ordering of the array elements in the array (6.2.2.2) or the array section (6.2.2.3). Every record of the file has the same length, which is the length of an array element in the array.

(4) A record of the internal file becomes defined by writing the record. If the number of characters written in a record is less than the length of the record, the remaining portion of the record is filled with blanks. The number of characters to be written shall not exceed the length of the record.

(5) A record may be read only if the record is defined.

(6) A record of an internal file may become defined (or undefined) by means other than an output statement. For example, the character variable may become defined by a character assignment statement.

(7) An internal file is always positioned at the beginning of the first record prior to data transfer. This record becomes the current record.

(8) On input, blanks are treated in the same way as for an external file opened with a BLANK= specifier having the value NULL and records are padded with blanks if necessary (9.4.4.4.2).

(9) On list-directed output, character constants are not delimited (10.8.2).

9.2.2.2 Internal file restrictions

An internal file has the following restrictions:

(1) Reading and writing records shall be accomplished only by sequential access formatted input/output statements that do not specify namelist formatting.

(2) An internal file shall not be specified in a file connection statement, a file positioning statement, or a file inquiry statement.

9.3 File connection

A **unit**, specified by an *io-unit*, provides a means for referring to a file.

R901	*io-unit*	**is**	*external-file-unit*
		or	*
		or	*internal-file-unit*
R902	*external-file-unit*	**is**	*scalar-int-expr*
R903	*internal-file-unit*	**is**	*default-char-variable*

Constraint: The *default-char-variable* shall not be an array section with a vector subscript.

A unit is either an external unit or an internal unit. An **external unit** is used to refer to an external file and is specified by an *external-file-unit* or an asterisk. An **internal unit** is used to refer to an internal file and is specified by an *internal-file-unit*.

If a character variable that identifies an internal file unit is a pointer, it shall be associated. If the character variable is an allocatable array or a subobject of such an array, the array shall be currently allocated.

A scalar integer expression that identifies an external file unit shall be zero or positive.

The *io-unit* in a file positioning statement, a file connection statement, or a file inquiry statement shall be an *external-file-unit*.

The external unit identified by the value of the *scalar-int-expr* is the same external unit in all program units of the program.

NOTE 9.8

In the example:

```
SUBROUTINE A
   READ (6) X
      ...
SUBROUTINE B
   N = 6
   REWIND N
```

the value 6 used in both program units identifies the same external unit.

An asterisk identifies particular processor-dependent external units that are preconnected for formatted sequential access (9.4.4.2).

9.3.1 Unit existence

At any given time, there is a processor-dependent set of external units that are said to exist for a program.

All input/output statements may refer to units that exist. The INQUIRE statement and the CLOSE statement also may refer to units that do not exist.

9.3.2 Connection of a file to a unit

An external unit has a property of being **connected** or not connected. If connected, it refers to an external file. An external unit may become connected by preconnection or by the execution of an

OPEN statement. The property of connection is symmetric; if a unit is connected to a file, the file is connected to the unit.

All input/output statements except an OPEN, a CLOSE, or an INQUIRE statement shall refer to a unit that is connected to a file and thereby make use of or affect that file.

A file may be connected and not exist (9.2.1.1).

NOTE 9.9

An example is a preconnected external file that has not yet been written.

A unit shall not be connected to more than one file at the same time, and a file shall not be connected to more than one unit at the same time. However, means are provided to change the status of an external unit and to connect a unit to a different file.

After an external unit has been disconnected by the execution of a CLOSE statement, it may be connected again within the same program to the same file or to a different file. After an external file has been disconnected by the execution of a CLOSE statement, it may be connected again within the same program to the same unit or to a different unit.

NOTE 9.10

The only means of referencing a file that has been disconnected is by the appearance of its name in an OPEN or INQUIRE statement. There may be no means of reconnecting an unnamed file once it is disconnected.

An internal unit is always connected to the internal file designated by the variable of default character type that identifies the unit.

NOTE 9.11

For more explanatory information on file connection properties, see C.6.3.

9.3.3 Preconnection

Preconnection means that the unit is connected to a file at the beginning of execution of the program and therefore it may be specified in input/output statements without the prior execution of an OPEN statement.

9.3.4 The OPEN statement

An **OPEN statement** initiates or modifies the connection between an external file and a specified unit. The OPEN statement may be used to connect an existing file to a unit, create a file that is preconnected, create a file and connect it to a unit, or change certain specifiers of a connection between a file and a unit.

An external unit may be connected by an OPEN statement in any program unit of a program and, once connected, a reference to it may appear in any program unit of the program.

If a unit is connected to a file that exists, execution of an OPEN statement for that unit is permitted. If the FILE= specifier is not included in such an OPEN statement, the file to be connected to the unit is the same as the file to which the unit is already connected.

If the file to be connected to the unit does not exist but is the same as the file to which the unit is preconnected, the properties specified by an OPEN statement become a part of the connection.

If the file to be connected to the unit is not the same as the file to which the unit is connected, the effect is as if a CLOSE statement without a STATUS= specifier had been executed for the unit immediately prior to the execution of an OPEN statement.

If the file to be connected to the unit is the same as the file to which the unit is connected, only the BLANK=, DELIM=, PAD=, ERR=, and IOSTAT= specifiers may have values different from those currently in effect. If the POSITION= specifier is present in such an OPEN statement, the value specified shall not disagree with the current position of the file. If the STATUS= specifier is included in such an OPEN statement, it shall be specified with a value of OLD. Execution of such an OPEN statement causes any new value of the BLANK=, DELIM=, or PAD= specifiers to be in effect, but does not cause any change in any of the unspecified specifiers and the position of the file is unaffected. The ERR= and IOSTAT= specifiers from any previously executed OPEN statement have no effect on any currently executed OPEN statement.

A STATUS= specifier with a value of OLD is always allowed when the file to be connected to the unit is the same as the file to which the unit is connected. In this case, if the status of the file was SCRATCH before execution of the OPEN statement, the file will still be deleted when the unit is closed, and the file is still considered to have a status of SCRATCH.

If a file is already connected to a unit, execution of an OPEN statement on that file and a different unit is not permitted.

R904	*open-stmt*	**is**	OPEN (*connect-spec-list*)
R905	*connect-spec*	**is**	[UNIT =] *external-file-unit*
		or	IOSTAT = *scalar-default-int-variable*
		or	ERR = *label*
		or	FILE = *file-name-expr*
		or	STATUS = *scalar-default-char-expr*
		or	ACCESS = *scalar-default-char-expr*
		or	FORM = *scalar-default-char-expr*
		or	RECL = *scalar-int-expr*
		or	BLANK = *scalar-default-char-expr*
		or	POSITION = *scalar-default-char-expr*
		or	ACTION = *scalar-default-char-expr*
		or	DELIM = *scalar-default-char-expr*
		or	PAD = *scalar-default-char-expr*
R906	*file-name-expr*	**is**	*scalar-default-char-expr*

Constraint: If the optional characters UNIT= are omitted from the unit specifier, the unit specifier shall be the first item in the *connect-spec-list*.

Constraint: Each specifier shall not appear more than once in a given *open-stmt*; an *external-file-unit* shall be specified.

Constraint: The *label* used in the ERR= specifier shall be the statement label of a branch target statement that appears in the same scoping unit as the OPEN statement.

If the STATUS= specifier has the value NEW or REPLACE, the FILE= specifier shall be present. If the STATUS= specifier has the value SCRATCH, the FILE= specifier shall be absent. If the STATUS= specifier has the value OLD, the FILE= specifier shall be present unless the unit is currently connected and the file connected to the unit exists.

A specifier that requires a *scalar-default-char-expr* may have a limited list of character values. These values are listed for each such specifier. Any trailing blanks are ignored. If a processor is capable of representing letters in both upper and lower case, the value specified is without regard to case. Some specifiers have a default value if the specifier is omitted.

The IOSTAT= specifier and ERR= specifier are described in 9.4.1.4 and 9.4.1.5, respectively.

NOTE 9.12

An example of an OPEN statement is:

```
OPEN (10, FILE = 'employee.names', ACTION = 'READ', PAD = 'YES')
```

NOTE 9.13

For more explanatory information on the OPEN statement, see C.6.2.

9.3.4.1 FILE= specifier in the OPEN statement

The value of the FILE= specifier is the name of the file to be connected to the specified unit. Any trailing blanks are ignored. The *file-name-expr* shall be a name that is allowed by the processor. If this specifier is omitted and the unit is not connected to a file, the STATUS= specifier shall be specified with a value of SCRATCH; in this case, the connection is made to a processor-dependent file. If a processor is capable of representing letters in both upper and lower case, the interpretation of case is processor dependent.

9.3.4.2 STATUS= specifier in the OPEN statement

The *scalar-default-char-expr* shall evaluate to OLD, NEW, SCRATCH, REPLACE, or UNKNOWN. If OLD is specified, the file shall exist. If NEW is specified, the file shall not exist.

Successful execution of an OPEN statement with NEW specified creates the file and changes the status to OLD. If REPLACE is specified and the file does not already exist, the file is created and the status is changed to OLD. If REPLACE is specified and the file does exist, the file is deleted, a new file is created with the same name, and the status is changed to OLD. If SCRATCH is specified, the file is created and connected to the specified unit for use by the program but is deleted at the execution of a CLOSE statement referring to the same unit or at the termination of the program.

NOTE 9.14

SCRATCH shall not be specified with a named file.

If UNKNOWN is specified, the status is processor dependent. If this specifier is omitted, the default value is UNKNOWN.

9.3.4.3 ACCESS= specifier in the OPEN statement

The *scalar-default-char-expr* shall evaluate to SEQUENTIAL or DIRECT. The ACCESS= specifier specifies the access method for the connection of the file as being sequential or direct. If this specifier is omitted, the default value is SEQUENTIAL. For an existing file, the specified access method shall be included in the set of allowed access methods for the file. For a new file, the processor creates the file with a set of allowed access methods that includes the specified method.

9.3.4.4 FORM= specifier in the OPEN statement

The *scalar-default-char-expr* shall evaluate to FORMATTED or UNFORMATTED. The FORM= specifier determines whether the file is being connected for formatted or unformatted input/output. If this specifier is omitted, the default value is UNFORMATTED if the file is being connected for direct access, and the default value is FORMATTED if the file is being connected for sequential access. For an existing file, the specified form shall be included in the set of allowed forms for the file. For a new file, the processor creates the file with a set of allowed forms that includes the specified form.

9.3.4.5 RECL= specifier in the OPEN statement

The value of the RECL= specifier shall be positive. It specifies the length of each record in a file being connected for direct access, or specifies the maximum length of a record in a file being connected for sequential access. This specifier shall be present when a file is being connected for direct access. If this specifier is omitted when a file is being connected for sequential access, the default value is processor dependent. If the file is being connected for formatted input/output, the length is the number of characters for all records that contain only characters of type default character. When a record contains any nondefault characters, the appropriate value for the RECL= specifier is processor dependent. If the file is being connected for unformatted input/output, the length is measured in processor-dependent units. For an existing file, the value of the RECL=

specifier shall be included in the set of allowed record lengths for the file. For a new file, the processor creates the file with a set of allowed record lengths that includes the specified value.

9.3.4.6 BLANK= specifier in the OPEN statement

The *scalar-default-char-expr* shall evaluate to NULL or ZERO. The BLANK= specifier is permitted only for a file being connected for formatted input/output. If NULL is specified, all blank characters in numeric formatted input fields on the specified unit are ignored, except that a field of all blanks has a value of zero. If ZERO is specified, all blanks other than leading blanks are treated as zeros. If this specifier is omitted, the default value is NULL.

9.3.4.7 POSITION= specifier in the OPEN statement

The *scalar-default-char-expr* shall evaluate to ASIS, REWIND, or APPEND. The connection shall be for sequential access. A file that did not exist previously (a new file, either specified explicitly or by default) is positioned at its initial point. REWIND positions an existing file at its initial point. APPEND positions an existing file such that the endfile record is the next record, if it has one. If an existing file does not have an endfile record, APPEND positions the file at its terminal point. ASIS leaves the position unchanged if the file exists and already is connected. ASIS leaves the position unspecified if the file exists but is not connected. If this specifier is omitted, the default value is ASIS.

9.3.4.8 ACTION= specifier in the OPEN statement

The *scalar-default-char-expr* shall evaluate to READ, WRITE, or READWRITE. READ specifies that the WRITE, PRINT, and ENDFILE statements shall not refer to this connection. WRITE specifies that READ statements shall not refer to this connection. READWRITE permits any I/O statements to refer to this connection. If this specifier is omitted, the default value is processor dependent. If READWRITE is included in the set of allowable actions for a file, both READ and WRITE also shall be included in the set of allowed actions for that file. For an existing file, the specified action shall be included in the set of allowed actions for the file. For a new file, the processor creates the file with a set of allowed actions that includes the specified action.

9.3.4.9 DELIM= specifier in the OPEN statement

The *scalar-default-char-expr* shall evaluate to APOSTROPHE, QUOTE, or NONE. If APOSTROPHE is specified, the apostrophe shall be used to delimit character values written with list-directed or namelist formatting and all internal apostrophes shall be doubled. If QUOTE is specified, the quotation mark shall be used to delimit character values written with list-directed or namelist formatting and all internal quotation marks shall be doubled. If the value of this specifier is NONE, a character value when written shall not be delimited by apostrophes or quotation marks, nor shall any internal apostrophes or quotation marks be doubled. If this specifier is omitted, the default value is NONE. This specifier is permitted only for a file being connected for formatted input/output. This specifier is ignored during input of a formatted record.

9.3.4.10 PAD= specifier in the OPEN statement

The *scalar-default-char-expr* shall evaluate to YES or NO. If YES is specified, a formatted input record is padded with blanks (9.4.4.4.2) when an input list is specified and the format specification requires more data from a record than the record contains. If NO is specified, the input list and the format specification shall not require more characters from a record than the record contains. If this specifier is omitted, the default value is YES. This specifier is permitted only for a file being connected for formatted input/output. This specifier is ignored during output of a formatted record.

NOTE 9.15

For nondefault character types, the blank padding character is processor dependent.

9.3.5 The CLOSE statement

The **CLOSE statement** is used to terminate the connection of a specified unit to an external file.

Execution of a CLOSE statement that refers to a unit may occur in any program unit of a program and need not occur in the same program unit as the execution of an OPEN statement referring to that unit.

Execution of a CLOSE statement specifying a unit that does not exist or has no file connected to it is permitted and affects no file.

After a unit has been disconnected by execution of a CLOSE statement, it may be connected again within the same program, either to the same file or to a different file. After a named file has been disconnected by execution of a CLOSE statement, it may be connected again within the same program, either to the same unit or to a different unit, provided that the file still exists.

At termination of execution of a program for reasons other than an error condition, all units that are connected are closed. Each unit is closed with status KEEP unless the file status prior to termination of execution was SCRATCH, in which case the unit is closed with status DELETE.

NOTE 9.16

The effect is as though a CLOSE statement without a STATUS= specifier were executed on each connected unit.

R907 *close-stmt* **is** CLOSE (*close-spec-list*)

R908 *close-spec* **is** [UNIT =] *external-file-unit*
or IOSTAT = *scalar-default-int-variable*
or ERR = *label*
or STATUS = *scalar-default-char-expr*

Constraint: If the optional characters UNIT= are omitted from the unit specifier, the unit specifier shall be the first item in the *close-spec-list*.

Constraint: Each specifier shall not appear more than once in a given *close-stmt*; an *external-file-unit* shall be specified.

Constraint: The *label* used in the ERR= specifier shall be the statement label of a branch target statement that appears in the same scoping unit as the CLOSE statement.

The *scalar-default-char-expr* has a limited list of character values. Any trailing blanks are ignored. If a processor is capable of representing letters in both upper and lower case, the value specified is without regard to case.

The IOSTAT= specifier and ERR= specifier are described in 9.4.1.4 and 9.4.1.5, respectively.

NOTE 9.17

An example of a CLOSE statement is:

```
CLOSE (10, STATUS = 'KEEP')
```

NOTE 9.18

For more explanatory information on the CLOSE statement, see C.6.4.

9.3.5.1 STATUS= specifier in the CLOSE statement

The *scalar-default-char-expr* shall evaluate to KEEP or DELETE. The STATUS= specifier determines the disposition of the file that is connected to the specified unit. KEEP shall not be specified for a file whose status prior to execution of a CLOSE statement is SCRATCH. If KEEP is specified for a file that exists, the file continues to exist after the execution of a CLOSE statement. If KEEP is specified for a file that does not exist, the file will not exist after the execution of a CLOSE statement. If DELETE is specified, the file will not exist after the execution of a CLOSE statement.

If this specifier is omitted, the default value is KEEP, unless the file status prior to execution of the CLOSE statement is SCRATCH, in which case the default value is DELETE.

9.4 Data transfer statements

The **READ statement** is the data transfer input statement. The **WRITE statement** and the **PRINT statement** are the data transfer output statements.

R909 *read-stmt* **is** READ (*io-control-spec-list*) [*input-item-list*]
or READ *format* [, *input-item-list*]

R910 *write-stmt* **is** WRITE (*io-control-spec-list*) [*output-item-list*]

R911 *print-stmt* **is** PRINT *format* [, *output-item-list*]

NOTE 9.19

Examples of data transfer statements are:

```
   READ (6, *) SIZE
   READ 10, A, B
   WRITE (6, 10) A, S, J
   PRINT 10, A, S, J
10 FORMAT (2E16.3, I5)
```

9.4.1 Control information list

The *io-control-spec-list* is a **control information list** that includes

(1) A reference to the source or destination of the data to be transferred,
(2) Optional specification of editing processes,
(3) Optional specification to identify a record,
(4) Optional specification of exception handling,
(5) Optional return of status,
(6) Optional record advancing specification, and
(7) Optional return of number of characters read.

The control information list governs the data transfer.

R912 *io-control-spec* **is** [UNIT =] *io-unit*
or [FMT =] *format*
or [NML =] *namelist-group-name*
or REC = *scalar-int-expr*
or IOSTAT = *scalar-default-int-variable*
or ERR = *label*
or END = *label*
or ADVANCE = *scalar-default-char-expr*
or SIZE = *scalar-default-int-variable*
or EOR = *label*

Constraint: An *io-control-spec-list* shall contain exactly one *io-unit* and may contain at most one of each of the other specifiers.

Constraint: An END=, EOR=, or SIZE= specifier shall not appear in a *write-stmt*.

Constraint: The *label* in the ERR=, EOR=, or END= specifier shall be the statement label of a branch target statement that appears in the same scoping unit as the data transfer statement.

Constraint: A *namelist-group-name* shall not be present if an *input-item-list* or an *output-item-list* is present in the data transfer statement.

Constraint: An *io-control-spec-list* shall not contain both a *format* and a *namelist-group-name*.

Constraint: If the optional characters UNIT= are omitted fro m the unit specifier, the unit specifier shall be the first item in the control information list.

Constraint: If the optional characters FMT= are omitted from the format specifier, the format specifier shall be the second item in the control information list and the first item shall be the unit specifier without the optional characters UNIT=.

Constraint: If the optional characters NML= are omitted from the namelist specifier, the namelist specifier shall be the second item in the control information list and the first item shall be the unit specifier without the optional characters UNIT=.

Constraint: If the unit specifier specifies an internal file, the *io-control-spec-list* shall not contain a REC= specifier or a *namelist-group-name*.

Constraint: If the REC= specifier is present, an END= specifier shall not appear, a *namelist-group-name* shall not appear, and the *format*, if any, shall not be an asterisk specifying list-directed input/output.

Constraint: An ADVANCE= specifier may be present only in a formatted sequential input/output statement with explicit format specification (10.1) whose control information list does not contain an internal file unit specifier.

Constraint: If an EOR= specifier is present, an ADVANCE= specifier also shall appear.

Constraint: If a SIZE= specifier is present, an ADVANCE= specifier also shall appear.

A SIZE= specifier may be present only in an input statement that contains an ADVANCE= specifier with the value NO.

An EOR= specifier may be present only in an input statement that contains an ADVANCE= specifier with the value NO.

If the data transfer statement contains a *format* or *namelist-group-name*, the statement is a **formatted input/output statement** ; otherwise, it is an **unformatted input/output statement**.

In a data transfer statement, the variable specified in an IOSTAT= or a SIZE= specifier, if any, shall not be associated with any entity in the data transfer input/output list (9.4.2) or *namelist-group-object-list*, nor with a *do-variable* of an *io-implied-do* in the data transfer input/output list.

In a data transfer statement, if a variable specified in an IOSTAT= or a SIZE= specifier is an array element reference, its subscript values shall not be affected by the data transfer, the *io-implied-do* processing, or the definition or evaluation of any other specifier in the *io-control-spec-list*.

For the ADVANCE= specifier, the *scalar-default-char-expr* has a limited list of character values. Any trailing blanks are ignored. If a processor is capable of representing letters in both upper and lower case, the value specified is without regard to case.

NOTE 9.20

An example of a READ statement is:

```
READ (IOSTAT = IOS, UNIT = 6, FMT = '(10F8.2)') A, B
```

9.4.1.1 Format specifier

The FMT= specifier supplies a format specification or specifies list-directed formatting for a formatted input/output statement.

R913 *format* **is** *default-char-expr*
or *label*
or *

Constraint: The *label* shall be the label of a FORMAT statement that appears in the same scoping unit as the statement containing the format specifier.

The *default-char-expr* shall evaluate to a valid format specification (10.1.1 and 10.1.2).

NOTE 9.21

A default-char-expr includes a character constant.

If *default-char-expr* is an array, it is treated as if all of the elements of the array were specified in array element order and were concatenated.

If *format* is ∗, the statement is a **list-directed input/output statement**.

NOTE 9.22

An example in which the format is a character expression is:

```
READ (6, FMT = "(" // CHAR_FMT // ")" )  X, Y, Z
```

where CHAR_FMT is a default character variable.

9.4.1.2 Namelist specifier

The NML= specifier supplies the *namelist-group-name* (5.4). This name identifies a specific collection of data objects on which transfer is to be performed.

If a *namelist-group-name* is present, the statement is a **namelist input/output statement**.

9.4.1.3 Record number

The REC= specifier specifies the number of the record that is to be read or written. This specifier may be present only in an input/output statement that specifies a unit connected for direct access. If the control information list contains a REC= specifier, the statement is a **direct access input/output statement** ; otherwise, it is a **sequential access input/output statement**.

9.4.1.4 Input/output status

Execution of an input/output statement containing the IOSTAT= specifier causes the variable specified in the IOSTAT= specifier to become defined

(1) With a zero value if neither an error condition, an end-of-file condition, nor an end-of-record condition occurs,

(2) With a processor-dependent positive integer value if an error condition occurs,

(3) With a processor-dependent negative integer value if an end-of-file condition occurs and no error condition occurs, or

(4) With a processor-dependent negative integer value different from the end-of-file value if an end-of-record condition occurs and no error condition or end-of-file condition occurs.

NOTE 9.23

An end-of-file condition may occur only during execution of a sequential input statement and an end-of-record condition may occur only during execution of a nonadvancing input statement.

Consider the example:

```
READ (FMT = "(E8.3)", UNIT = 3, IOSTAT = IOSS) X
IF (IOSS < 0) THEN
   ! Perform end-of-file processing on the file connected to unit 3.
   CALL END_PROCESSING
ELSE IF (IOSS > 0) THEN
   ! Perform error processing
   CALL ERROR_PROCESSING
END IF
```

9.4.1.5 Error branch

If an error condition (9.4.3) occurs during execution of an input/output statement that contains an ERR= specifier

(1) Execution of the input/output statement terminates,

(2) The position of the file specified in the input/output statement becomes indeterminate,

(3) If the input/output statement also contains an IOSTAT= specifier, the variable specified becomes defined with a processor-dependent positive integer value,

(4) If the statement is a READ statement and it contains a SIZE= specifier, the variable becomes defined with an integer value (9.4.1.9), and

(5) Execution continues with the statement specified in the ERR= specifier.

9.4.1.6 End-of-file branch

If an end-of-file condition (9.4.3) occurs and no error condition (9.4.3) occurs during execution of an input statement that contains an END= specifier

(1) Execution of the input statement terminates,

(2) If the file specified in the input statement is an external file, it is positioned after the endfile record,

(3) If the input statement also contains an IOSTAT= specifier, the variable specified becomes defined with a processor-dependent negative integer value, and

(4) Execution continues with the statement specified in the END= specifier.

9.4.1.7 End-of-record branch

If an end-of-record condition (9.4.3) occurs and no error condition (9.4.3) occurs during execution of an input/output statement that contains an EOR= specifier

(1) If the PAD= specifier has the value YES, the record is padded with blanks to satisfy the input list item (9.4.4.4.2) and corresponding data edit descriptor that requires more characters than the record contains,

(2) Execution of the input statement terminates,

(3) The file specified in the input statement is positioned after the current record,

(4) If the input statement also contains an IOSTAT= specifier, the variable specified becomes defined with a processor-dependent negative integer value,

(5) If the input statement contains a SIZE= specifier, the variable becomes defined with an integer value (9.4.1.9), and

(6) Execution continues with the statement specified in the EOR= specifier.

9.4.1.8 Advance specifier

The *scalar-default-char-expr* shall evaluate to YES or NO. The ADVANCE= specifier determines whether nonadvancing input/output occurs for this input/output statement. If NO is specified, nonadvancing input/output occurs. If YES is specified, advancing formatted sequential input/output occurs. If this specifier is omitted, the default value is YES.

9.4.1.9 Character count

When a nonadvancing input statement terminates, the variable specified in the SIZE= specifier becomes defined with the count of the characters transferred by data edit descriptors during execution of the current input statement. Blanks inserted as padding (9.4.4.4.2) are not counted.

9.4.2 Data transfer input/output list

An input/output list specifies the entities whose values are transferred by a data transfer input/output statement.

R914 *input-item* **is** *variable*
or *io-implied-do*

R915 *output-item* **is** *expr*
or *io-implied-do*

R916 *io-implied-do* **is** (*io-implied-do-object-list* , *io-implied-do-control*)

R917 *io-implied-do-object* **is** *input-item*
or *output-item*

R918 *io-implied-do-control* **is** *do-variable* = *scalar-int-expr* , ■
■ *scalar-int-expr* [, *scalar-int-expr*]

Constraint: A variable that is an *input-item* shall not be a whole assumed-size array.

Constraint: The *do-variable* shall be a named scalar variable of type integer.

Constraint: In an *input-item-list*, an *io-implied-do-object* shall be an *input-item*. In an *output-item-list*, an *io-implied-do-object* shall be an *output-item*.

An *input-item* shall not appear as, nor be associated with, the *do-variable* of any *io-implied-do* that contains the *input-item*.

If an input item is a pointer, it shall be currently associated with a definable target and data are transferred from the file to the associated target. If an output item is a pointer, it shall be currently associated with a target and data are transferred from the target to the file.

NOTE 9.24

Data transfers always involve the movement of values between a file and internal storage. A pointer as such cannot be read or written. A pointer may, therefore, appear as an item in an input/output list if it is currently associated with a target that can receive a value (input) or can deliver a value (output). A derived-type object with one or more pointer components shall not appear as an item in an input/output list because the value of a pointer component is a descriptor for a location in memory. As such, this has no processor-independent representation.

If an input item or an output item is an allocatable array, it shall be currently allocated.

The *do-variable* of an *io-implied-do* that is in another *io-implied-do* shall not appear as, nor be associated with, the *do-variable* of the containing *io-implied-do*.

If an array appears as an input/output list item, it is treated as if the elements, if any, were specified in array element order (6.2.2.2). However, no element of that array may affect the value of any expression in the *input-item*, nor may any element appear more than once in an *input-item*.

NOTE 9.25

For example:

```
INTEGER A (100), J (100)
   ...
READ *, A (A)                                ! Not allowed
READ *, A (LBOUND (A, 1) : UBOUND (A, 1))    ! Allowed
READ *, A (J)                                ! Allowed if no two elements
                                             !    of J have the same value
READ *, A (A (1) : A (10))                   ! Not allowed
```

A derived-type object shall not appear as an input/output list item if any component ultimately in the object is not accessible within the scoping unit containing the input/output statement.

NOTE 9.26

An example is a structure accessed from a module within which its type is PUBLIC but its components are PRIVATE.

If a derived type ultimately contains a pointer component, an object of this type shall not appear as an input item nor as the result of the evaluation of an output list item.

If a derived-type object appears as an input/output list item in a formatted input/output statement, it is treated as if all of the components of the object were specified in the same order as in the definition of the derived type.

NOTE 9.27

In a formatted input/output statement, edit descriptors are associated with effective list items, which are always scalar and of intrinsic type. The rules in 9.4.2 determine the set of effective list items corresponding to each actual list item in the statement. These rules may have to be applied repetitively until all of the effective list items are scalar items of intrinsic type.

An input/output list item of derived type in an unformatted input/output statement is treated as a single value in a processor-dependent form.

NOTE 9.28

The appearance of a derived-type object as an input/output list item in an unformatted input/output statement is not equivalent to the list of its components.

Unformatted input/output involving derived-type list items forms the single exception to the rule that the appearance of an aggregate list item (such as an array) is equivalent to the appearance of its expanded list of component parts. This exception permits the processor greater latitude in improving efficiency or in matching the processor-dependent sequence of values for a derived-type object to similar sequences for aggregate objects used by means other than Fortran. However, formatted input/output of all list items and unformatted input/output of list items other than those of derived types adhere to the above rule.

For an implied-DO, the loop initialization and execution is the same as for a DO construct (8.1.4.4).

An input/output list shall not contain an item of nondefault character type if the input/output statement specifies an internal file.

NOTE 9.29

A constant, an expression involving operators or function references, or an expression enclosed in parentheses may appear as an output list item but shall not appear as an input list item.

NOTE 9.30

An example of an output list with an implied-DO is:

```
WRITE (LP, FMT = '(10F8.2)') (LOG (A (I)), I = 1, N + 9, K), G
```

9.4.3 Error, end-of-record, and end-of-file conditions

The set of input/output error conditions is processor dependent.

An **end-of-record condition** occurs when a nonadvancing input statement attempts to transfer data from a position beyond the end of the current record.

An **end-of-file condition** occurs in either of the following cases:

(1) When an endfile record is encountered during the reading of a file connected for sequential access.

(2) When an attempt is made to read a record beyond the end of an internal file.

An end-of-file condition may occur at the beginning of execution of an input statement. An end-of-file condition also may occur during execution of a formatted input statement when more than one record is required by the interaction of the input list and the format.

If an error condition or an end-of-file condition occurs during execution of an input/output statement, execution of the input/output statement terminates and if the input/output statement contains any implied-DOs, all of the implied-DO variables in the statement become undefined. If an error condition occurs during execution of an input/output statement, the position of the file becomes indeterminate.

If an error or end-of-file condition occurs on input, all input list items become undefined.

If an end-of-record condition occurs during execution of a nonadvancing input statement, the following occurs: if the PAD= specifier has the value YES, the record is padded with blanks (9.4.4.4.2) to satisfy the input list item and corresponding data edit descriptor that require more characters than the record contains; execution of the input statement terminates and if the input statement contains any implied-DOs, all of the implied-DO variables in the statement become undefined; and the file specified in the input statement is positioned after the current record.

Execution of the program is terminated if an error condition occurs during execution of an input/output statement that contains neither an IOSTAT= nor an ERR= specifier, or if an end-of-file condition occurs during execution of a READ statement that contains neither an IOSTAT= specifier nor an END= specifier, or if an end-of-record condition occurs during execution of a nonadvancing READ statement that contains neither an IOSTAT= specifier nor an EOR= specifier.

9.4.4 Execution of a data transfer input/output statement

The effect of executing a data transfer input/output statement shall be as if the following operations were performed in the order specified:

(1) Determine the direction of data transfer

(2) Identify the unit

(3) Establish the format if one is specified

(4) Position the file prior to data transfer (9.2.1.3.2)

(5) Transfer data between the file and the entities specified by the input/output list (if any) or namelist

(6) Determine whether an error condition, an end-of-file condition, or an end-of-record condition has occurred

(7) Position the file after data transfer (9.2.1.3.3)

(8) Cause any variables specified in the IOSTAT= and SIZE= specifiers to become defined.

9.4.4.1 Direction of data transfer

Execution of a READ statement causes values to be transferred from a file to the entities specified by the input list, if any, or specified within the file itself for namelist input. Execution of a WRITE or PRINT statement causes values to be transferred to a file from the entities specified by the output list and format specification, if any, or by the *namelist-group-name* for namelist output. Execution of a WRITE or PRINT statement for a file that does not exist creates the file unless an error condition occurs.

9.4.4.2 Identifying a unit

A data transfer input/output statement that contains an input/output control list includes a unit specifier that identifies an external unit or an internal file. A READ statement that does not contain an input/output control list specifies a particular processor-dependent unit, which is the same as the unit identified by ∗ in a READ statement that contains an input/output control list. The PRINT statement specifies some other processor-dependent unit, which is the same as the unit identified by ∗ in a WRITE statement. Thus, each data transfer input/output statement identifies an external unit or an internal file.

The unit identified by a data transfer input/output statement shall be connected to a file when execution of the statement begins.

NOTE 9.31

The file may be preconnected.

9.4.4.3 Establishing a format

If the input/output control list contains ∗ as a format, list-directed formatting is established. If *namelist-group-name* is present, namelist formatting is established. If no *format* or *namelist-group-name* is specified, unformatted data transfer is established. Otherwise, the format specification identified by the format specifier is established. If the format is an array, the effect is as if all elements of the array were concatenated in array element order.

On output, if an internal file has been specified, a format specification that is in the file or is associated with the file shall not be specified.

9.4.4.4 Data transfer

Data are transferred between records and entities specified by the input/output list or namelist. The list items are processed in the order of the input/output list for all data transfer input/output statements except namelist formatted data transfer statements. The next item to be processed in the list is called the **next effective item**. Zero-sized arrays and implied-DO lists with iteration counts of zero are ignored in determining the next effective item. A scalar character item of zero character length is treated as an effective item. The list items for a namelist input statement are processed in the order of the entities specified within the input records. The list items for a namelist output statement are processed in the order in which the data objects (variables) are specified in the *namelist-group-object-list*.

All values needed to determine which entities are specified by an input/output list item are determined at the beginning of the processing of that item.

All values are transmitted to or from the entities specified by a list item prior to the processing of any succeeding list item for all data transfer input/output statements.

NOTE 9.32

In the example,

```
READ (N) N, X (N)
```

the old value of N identifies the unit, but the new value of N is the subscript of X.

All values following the *name*= part of the namelist entity (10.9) within the input records are transmitted to the matching entity specified in the *namelist-group-object-list* prior to processing any succeeding entity within the input record for namelist input statements. If an entity is specified more than once within the input record during a namelist formatted data transfer input statement, the last occurrence of the entity specifies the value or values to be used for that entity.

An input list item, or an entity associated with it, shall not contain any portion of an established format specification.

If the input/output item is a pointer, data are transferred between the file and the associated target.

If an internal file has been specified, an input/output list item shall not be in the file or associated with the file.

NOTE 9.33

The file is a data object.

A DO variable becomes defined and its iteration count established at the beginning of processing of the items that constitute the range of an *io-implied-do*.

On output, every entity whose value is to be transferred shall be defined.

9.4.4.4.1 Unformatted data transfer

During unformatted data transfer, data are transferred without editing between the current record and the entities specified by the input/output list. Exactly one record is read or written.

Objects of intrinsic or derived types may be transferred by means of an unformatted data transfer statement.

On input, the file shall be positioned so that the record read is an unformatted record or an endfile record. The number of values required by the input list shall be less than or equal to the number of values in the record. Each value in the record shall be of the same type as the corresponding entity in the input list, except that one complex value may correspond to two real list entities or two real values may correspond to one complex list entity. The type parameters of the corresponding entities shall be the same.

NOTE 9.34

If an entity in the input list is of type character, the character entity shall have the same length and the same kind type parameter as the character value. Also, if two real values correspond to one complex entity or one complex value corresponds to two real entities, all three shall have the same kind type parameter value.

On output to a file connected for unformatted direct access, the output list shall not specify more values than can fit into the record. If the file is connected for direct access and the values specified by the output list do not fill the record, the remainder of the record is undefined.

If the file is connected for unformatted sequential access, the record is created with a length sufficient to hold the values from the output list. This length shall be one of the set of allowed record lengths for the file and shall not exceed the value specified in the RECL= specifier, if any, of the OPEN statement that established the connection.

If the file is connected for formatted input/output, unformatted data transfer is prohibited.

The unit specified shall be an external unit.

9.4.4.4.2 Formatted data transfer

During formatted data transfer, data are transferred with editing between the file and the entities specified by the input/output list or by the *namelist-group-name*, if any. Format control is initiated and editing is performed as described in Section 10. The current record and possibly additional records are read or written.

Values may be transferred by means of a formatted data transfer statement to or from objects of intrinsic or derived types. In the latter case, the transfer is in the form of values of intrinsic types to or from the components of intrinsic types that ultimately comprise these structured objects.

On input, the file shall be positioned so that the record read is a formatted record or an endfile record.

If the file is connected for unformatted input/output, formatted data transfer is prohibited.

During advancing input from a file whose PAD= specifier has the value NO, the input list and format specification shall not require more characters from the record than the record contains.

During advancing input from a file whose PAD= specifier has the value YES, or during input from an internal file, blank characters are supplied by the processor if the input list and format specification require more characters from the record than the record contains.

During nonadvancing input from a file whose PAD= specifier has the value NO, an end-of-record condition (9.4.3) occurs if the input list and format specification require more characters from the record than the record contains.

During nonadvancing input from a file whose PAD= specifier has the value YES, an end-of-record condition occurs and blank characters are supplied by the processor if an input item and its corresponding data edit descriptor require more characters from the record than the record contains.

If the file is connected for direct access, the record number is increased by one as each succeeding record is read or written.

On output, if the file is connected for direct access or is an internal file and the characters specified by the output list and format do not fill a record, blank characters are added to fill the record.

On output, the output list and format specification shall not specify more characters for a record than have been specified by a RECL= specifier in the OPEN statement or the record length of an internal file.

9.4.4.5 List-directed formatting

If list-directed formatting has been established, editing is performed as described in 10.8.

9.4.4.6 Namelist formatting

If namelist formatting has been established, editing is performed as described in 10.9.

9.4.5 Printing of formatted records

The transfer of information in a formatted record to certain devices determined by the processor is called **printing**. If a formatted record is printed, the first character of the record is not printed. The remaining characters of the record, if any, are printed in one line beginning at the left margin.

The first character of such a record shall be of default character type and determines vertical spacing as follows:

Character	Vertical spacing before printing
Blank	One line
0	Two lines
1	To first line of next page
+	No advance

If there are no characters in the record, the vertical spacing is one line and no characters other than blank are printed in that line.

The PRINT statement does not imply that printing will occur, and the WRITE statement does not imply that printing will not occur.

9.4.6 Termination of data transfer statements

Termination of an input/output data transfer statement occurs when any of the following conditions are met:

(1) Format processing encounters a data edit descriptor and there are no remaining elements in the *input-item-list* or *output-item-list*.

(2) Unformatted or list-directed data transfer exhausts the *input-item-list* or *output-item-list*.

(3) Namelist output exhausts the *namelist-group-object-list*.

(4) An error condition occurs.

(5) An end-of-file condition occurs.

(6) A slash (/) is encountered as a value separator (10.8, 10.9) in the record being read during list-directed or namelist input.

(7) An end-of-record condition occurs during execution of a nonadvancing input statement (9.4.3).

9.5 File positioning statements

R919 *backspace-stmt* **is** BACKSPACE *external-file-unit*
or BACKSPACE (*position-spec-list*)

R920 *endfile-stmt* **is** ENDFILE *external-file-unit*
or ENDFILE (*position-spec-list*)

R921 *rewind-stmt* **is** REWIND *external-file-unit*
or REWIND (*position-spec-list*)

A file that is not connected for sequential access shall not be referred to by a BACKSPACE, an ENDFILE, or a REWIND statement. A file that is connected with an ACTION= specifier having the value READ shall not be referred to by an ENDFILE statement.

R922 *position-spec* **is** [UNIT =] *external-file-unit*
or IOSTAT = *scalar-default-int-variable*
or ERR = *label*

Constraint: The *label* in the ERR= specifier shall be the statement label of a branch target statement that appears in the same scoping unit as the file positioning statement.

Constraint: If the optional characters UNIT= are omitted from the unit specifier, the unit specifier shall be the first item in the *position-spec-list*.

Constraint: A *position-spec-list* shall contain exactly one *external-file-unit* and may contain at most one of each of the other specifiers.

The IOSTAT= and ERR= specifiers are described in 9.4.1.4 and 9.4.1.5, respectively.

9.5.1 BACKSPACE statement

Execution of a BACKSPACE statement causes the file connected to the specified unit to be positioned before the current record if there is a current record, or before the preceding record if there is no current record. If there is no current record and no preceding record, the position of the file is not changed.

NOTE 9.35

If the preceding record is an endfile record, the file is positioned before the endfile record.

If a BACKSPACE statement causes the implicit writing of an endfile record, the file is positioned before the record that precedes the endfile record.

Backspacing a file that is connected but does not exist is prohibited.

Backspacing over records written using list-directed or namelist formatting is prohibited.

NOTE 9.36

An example of a BACKSPACE statement is:

```
BACKSPACE (10, ERR = 20)
```

9.5.2 ENDFILE statement

Execution of an ENDFILE statement writes an endfile record as the next record of the file. The file is then positioned after the endfile record which becomes the last record of the file. If the file also may be connected for direct access, only those records before the endfile record are considered to have been written. Thus, only those records may be read during subsequent direct access connections to the file.

After execution of an ENDFILE statement, a BACKSPACE or REWIND statement shall be used to reposition the file prior to execution of any data transfer input/output statement or ENDFILE statement.

Execution of an ENDFILE statement for a file that is connected but does not exist creates the file prior to writing the endfile record.

NOTE 9.37

An example of an ENDFILE statement is:

```
ENDFILE K
```

9.5.3 REWIND statement

Execution of a REWIND statement causes the specified file to be positioned at its initial point.

NOTE 9.38

If the file is already positioned at its initial point, execution of this statement has no effect on the position of the file.

Execution of a REWIND statement for a file that is connected but does not exist is permitted and has no effect.

NOTE 9.39

An example of a REWIND statement is:

```
REWIND 10
```

9.6 File inquiry

The INQUIRE statement may be used to inquire about properties of a particular named file or of the connection to a particular unit. There are three forms of the INQUIRE statement: **inquire by file**, which uses the FILE= specifier, **inquire by unit**, which uses the UNIT= specifier, and **inquire by output list**, which uses only the IOLENGTH= specifier. All specifier value assignments are performed according to the rules for assignment statements.

An INQUIRE statement may be executed before, while, or after a file is connected to a unit. All values assigned by an INQUIRE statement are those that are current at the time the statement is executed.

R923	*inquire-stmt*	**is**	INQUIRE (*inquire-spec-list*)
		or	INQUIRE (IOLENGTH = *scalar-default-int-variable*) ■
			■ *output-item-list*

NOTE 9.40

Examples of INQUIRE statements are:

```
INQUIRE (IOLENGTH = IOL) A (1:N)
INQUIRE (UNIT = JOAN, OPENED = LOG_01, NAMED = LOG_02, &
   FORM = CHAR_VAR, IOSTAT = IOS)
```

NOTE 9.41

For more explanatory information on the INQUIRE statement, see C.6.5.

9.6.1 Inquiry specifiers

Unless constrained, the following inquiry specifiers may be used in either of the inquire by file or inquire by unit forms of the INQUIRE statement:

R924 *inquire-spec* **is** [UNIT =] *external-file-unit*
or FILE = *file-name-expr*
or IOSTAT = *scalar-default-int-variable*
or ERR = *label*
or EXIST = *scalar-default-logical-variable*
or OPENED = *scalar-default-* *variable*
or NUMBER = *scalar-default-int-variable*
or NAMED = *scalar-default-logical-variable*
or NAME = *scalar-default-char-variable*
or ACCESS = *scalar-default-char-variable*
or SEQUENTIAL = *scalar-default-char-variable*
or DIRECT = *scalar-default-char-variable*
or FORM = *scalar-default-char-variable*
or FORMATTED = *scalar-default-char-variable*
or UNFORMATTED = *scalar-default-char-variable*
or RECL = *scalar-default-int-variable*
or NEXTREC = *scalar-default-int-variable*
or BLANK = *scalar-default-char-variable*
or POSITION = *scalar-default-char-variable*
or ACTION = *scalar-default-char-variable*
or READ = *scalar-default-char-variable*
or WRITE = *scalar-default-char-variable*
or READWRITE = *scalar-default-char-variable*
or DELIM = *scalar-default-char-variable*
or PAD = *scalar-default-char-variable*

Constraint: An *inquire-spec-list* shall contain one FILE= specifier or one UNIT= specifier, but not both, and at most one of each of the other specifiers.

Constraint: In the inquire by unit form of the INQUIRE statement, if the optional characters UNIT= are omitted from the unit specifier, the unit specifier shall be the first item in the *inquire-spec-list*.

When a returned value of a specifier other than the NAME= specifier is of type character and the processor is capable of representing letters in both upper and lower case, the value returned is in upper case.

If an error condition occurs during execution of an INQUIRE statement, all of the inquiry specifier variables become undefined, except for the variable in the IOSTAT= specifier (if any).

The IOSTAT= and ERR= specifiers are described in 9.4.1.4 and 9.4.1.5, respectively.

9.6.1.1 FILE= specifier in the INQUIRE statement

The value of the *file-name-expr* in the FILE= specifier specifies the name of the file being inquired about. The named file need not exist or be connected to a unit. The value of the *file-name-expr* shall be of a form acceptable to the processor as a file name. Any trailing blanks are ignored. If a processor is capable of representing letters in both upper and lower case, the interpretation of case is processor dependent.

9.6.1.2 EXIST= specifier in the INQUIRE statement

Execution of an INQUIRE by file statement causes the *scalar-default-logical-variable* in the EXIST= specifier to be assigned the value true if there exists a file with the specified name; otherwise, false is assigned. Execution of an INQUIRE by unit statement causes true to be assigned if the specified unit exists; otherwise, false is assigned.

9.6.1.3 OPENED= specifier in the INQUIRE statement

Execution of an INQUIRE by file statement causes the *scalar-default-logical-variable* in the OPENED= specifier to be assigned the value true if the file specified is connected to a unit; otherwise, false is assigned. Execution of an INQUIRE by unit statement causes the *scalar-default-logical-variable* to be assigned the value true if the specified unit is connected to a file; otherwise, false is assigned.

9.6.1.4 NUMBER= specifier in the INQUIRE statement

The *scalar-default-int-variable* in the NUMBER= specifier is assigned the value of the external unit identifier of the unit that is currently connected to the file. If there is no unit connected to the file, the value –1 is assigned.

9.6.1.5 NAMED= specifier in the INQUIRE statement

The *scalar-default-logical-variable* in the NAMED= specifier is assigned the value true if the file has a name; otherwise, it is assigned the value false.

9.6.1.6 NAME= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the NAME= specifier is assigned the value of the name of the file if the file has a name; otherwise, it becomes undefined.

NOTE 9.42

If this specifier appears in an INQUIRE by file statement, its value is not necessarily the same as the name given in the FILE= specifier. However, the value returned shall be suitable for use as the value of the *file-name-expr* in the FILE= specifier in an OPEN statement.

The processor may return a file name qualified by a user identification, device, directory, or other relevant information.

If a processor is capable of representing letters in both upper and lower case, the case of the characters assigned to *scalar-default-char-variable* is processor dependent.

9.6.1.7 ACCESS= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the ACCESS= specifier is assigned the value SEQUENTIAL if the file is connected for sequential access, and DIRECT if the file is connected for direct access. If there is no connection, it is assigned the value UNDEFINED.

9.6.1.8 SEQUENTIAL= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the SEQUENTIAL= specifier is assigned the value YES if SEQUENTIAL is included in the set of allowed access methods for the file, NO if SEQUENTIAL is not included in the set of allowed access methods for the file, and UNKNOWN if the processor is unable to determine whether or not SEQUENTIAL is included in the set of allowed access methods for the file.

9.6.1.9 DIRECT= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the DIRECT= specifier is assigned the value YES if DIRECT is included in the set of allowed access methods for the file, NO if DIRECT is not included in the set of allowed access methods for the file, and UNKNOWN if the processor is unable to determine whether or not DIRECT is included in the set of allowed access methods for the file.

9.6.1.10 FORM= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the FORM= specifier is assigned the value FORMATTED if the file is connected for formatted input/output, and is assigned the value UNFORMATTED if the file is connected for unformatted input/output. If there is no connection, it is assigned the value UNDEFINED.

9.6.1.11 FORMATTED= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the FORMATTED= specifier is assigned the value YES if FORMATTED is included in the set of allowed forms for the file, NO if FORMATTED is not included in the set of allowed forms for the file, and UNKNOWN if the processor is unable to determine whether or not FORMATTED is included in the set of allowed forms for the file.

9.6.1.12 UNFORMATTED= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the UNFORMATTED= specifier is assigned the value YES if UNFORMATTED is included in the set of allowed forms for the file, NO if UNFORMATTED is not included in the set of allowed forms for the file, and UNKNOWN if the processor is unable to determine whether or not UNFORMATTED is included in the set of allowed forms for the file.

9.6.1.13 RECL= specifier in the INQUIRE statement

The *scalar-default-int-variable* in the RECL= specifier is assigned the value of the record length of a file connected for direct access, or the value of the maximum record length for a file connected for sequential access. If the file is connected for formatted input/output, the length is the number of characters for all records that contain only characters of type default character. If the file is connected for unformatted input/output, the length is measured in processor-dependent units. If there is no connection, the *scalar-default-int-variable* becomes undefined.

9.6.1.14 NEXTREC= specifier in the INQUIRE statement

The *scalar-default-int-variable* in the NEXTREC= specifier is assigned the value $n + 1$, where n is the record number of the last record read or written on the file connected for direct access. If the file is connected but no records have been read or written since the connection, the *scalar-default-int-variable* is assigned the value 1. If the file is not connected for direct access or if the position of the file is indeterminate because of a previous error condition, the *scalar-default-int-variable* becomes undefined.

9.6.1.15 BLANK= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the BLANK= specifier is assigned the value NULL if null blank control is in effect for the file connected for formatted input/output, and is assigned the value ZERO if zero blank control is in effect for the file connected for formatted input/output. If there is no connection, or if the connection is not for formatted input/output, the *scalar-default-char-variable* is assigned the value UNDEFINED.

9.6.1.16 POSITION= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the POSITION= specifier is assigned the value REWIND if the file is connected by an OPEN statement for positioning at its initial point, APPEND if the file is connected for positioning before its endfile record or at its terminal point, and ASIS if the file is connected without changing its position. If there is no connection or if the file is connected for direct access, the *scalar-default-char-variable* is assigned the value UNDEFINED. If the file has been repositioned since the connection, the *scalar-default-char-variable* is assigned a processor-dependent value, which shall not be REWIND unless the file is positioned at its initial point and shall not be APPEND unless the file is positioned so that its endfile record is the next record or at its terminal point if it has no endfile record.

9.6.1.17 ACTION= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the ACTION= specifier is assigned the value READ if the file is connected for input only, WRITE if the file is connected for output only, and READWRITE if it is connected for both input and output. If there is no connection, the *scalar-default-char-variable* is assigned the value UNDEFINED.

9.6.1.18 READ= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the READ= specifier is assigned the value YES if READ is included in the set of allowed actions for the file, NO if READ is not included in the set of allowed actions for the file, and UNKNOWN if the processor is unable to determine whether or not READ is included in the set of allowed actions for the file.

9.6.1.19 WRITE= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the WRITE= specifier is assigned the value YES if WRITE is included in the set of allowed actions for the file, NO if WRITE is not included in the set of allowed actions for the file, and UNKNOWN if the processor is unable to determine whether or not WRITE is included in the set of allowed actions for the file.

9.6.1.20 READWRITE= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the READWRITE= specifier is assigned the value YES if READWRITE is included in the set of allowed actions for the file, NO if READWRITE is not included in the set of allowed actions for the file, and UNKNOWN if the processor is unable to determine whether or not READWRITE is included in the set of allowed actions for the file.

9.6.1.21 DELIM= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the DELIM= specifier is assigned the value APOSTROPHE if the apostrophe is to be used to delimit character data written by list-directed or namelist formatting. If the quotation mark is used to delimit such data, the value QUOTE is assigned. If neither the apostrophe nor the quote is used to delimit the character data, the value NONE is assigned. If there is no connection or if the connection is not for formatted input/output, the *scalar-default-char-variable* is assigned the value UNDEFINED.

9.6.1.22 PAD= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the PAD= specifier is assigned the value NO if the connection of the file to the unit included the PAD= specifier and its value was NO. Otherwise, the *scalar-default-char-variable* is assigned the value YES.

9.6.2 Restrictions on inquiry specifiers

A variable that may become defined or undefined as a result of its use in a specifier in an INQUIRE statement, or any associated entity, shall not appear in another specifier in the same INQUIRE statement.

The *inquire-spec-list* in an INQUIRE by file statement shall contain exactly one FILE= specifier and shall not contain a UNIT= specifier. The *inquire-spec-list* in an INQUIRE by unit statement shall contain exactly one UNIT= specifier and shall not contain a FILE= specifier. The unit specified need not exist or be connected to a file. If it is connected to a file, the inquiry is being made about the connection and about the file connected.

9.6.3 Inquire by output list

The inquire by output list form of the INQUIRE statement does not include a FILE= or UNIT= specifier, and includes only an IOLENGTH= specifier and an output list.

The *scalar-default-int-variable* in the IOLENGTH= specifier is assigned the processor-dependent value that would result from the use of the output list in an unformatted output statement. The value shall be suitable as a RECL= specifier in an OPEN statement that connects a file for unformatted direct access when there are input/output statements with the same input/output list.

9.7 Restrictions on function references and list items

A function reference shall not appear in an expression anywhere in an input/output statement if such a reference causes another input/output statement to be executed.

NOTE 9.43

Restrictions in the evaluation of expressions (7.1.7) prohibit certain side effects.

9.8 Restriction on input/output statements

If a unit, or a file connected to a unit, does not have all of the properties required for the execution of certain input/output statements, those statements shall not refer to the unit.