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, WAIT, 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**.

A file is composed of either a sequence of file storage units or a sequence of records, which provide an extra level of organization to the file. A file composed of records is called a **record file**. A file composed of file storage units is called a **stream file**. A processor may allow a file to be viewed both as a record file and as a stream file; in this case the relationship between the file storage units when viewed as a stream file and the records when viewed as a record file is processor dependent.

A file is either an external file or an internal file.

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 file storage units (9.2.4) and depends on the output list (9.5.2) 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 (9.7.1).

9.2 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 9.3

An example of a restriction on input/output statements (9.10) 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 (9.2.3).

NOTE 9.4

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

9.2.1 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.

9.2.2 File access

There are three methods of accessing the data of an external file: sequential, direct, and stream. 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 (9.4.3) or when the file is created if the file is preconnected (9.4.4).

9.2.2.1 Sequential access

Sequential access is a method of accessing the records of an external record file in the order of their record numbers.

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 be read or written only by sequential access input/output statements.

9.2.2.2 Direct access

Direct access is a method of accessing the records of an external record file in arbitrary order.

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) The records of the file shall be read or written 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, and if a READ statement for this connection is permitted.

(6) The records of the file shall not be read or written using list-directed formatting (10.9), namelist formatting (10.10), or a nonadvancing input/output statement (9.2.3.1).

9.2.2.3 Stream access

Stream access is a method of accessing the file storage units (9.2.4) of an external stream file.

The properties of an external file connected for stream access depend on whether the connection is for unformatted or formatted access.

When connected for unformatted stream access, an external file has the following properties:

(1) The file storage units of the file shall be read or written only by stream access input/output statements.

(2) Each file storage unit in the file is uniquely identified by a positive integer called the position. The first file storage unit in the file is at position 1. The position of each subsequent file storage unit is one greater than that of its preceding unit.

(3) File storage units need not be read or written in order of their position. For example, it might be permissible to write the file storage unit at position 3, even though the file storage units at positions 1 and 2 have not been written. Any file storage unit may be read from the file while it is connected to a unit, provided that the file storage unit has been written since the file was created, and if a READ statement for this connection is permitted.

When connected for formatted stream access, an external file has the following properties:

(1) Some file storage units of the file may contain record markers; this imposes a record structure on the file in addition to its stream structure. There may or may not be a record marker at the end of the file. If there is no record marker at the end of the file, the final record is incomplete.

(2) No maximum length (9.4.5.11) is applicable to these records.

(3) Writing an empty record with no record marker has no effect.

NOTE 9.8

Because the record structure is determined from the record markers that are stored in the file itself, an incomplete record at the end of the file is necessarily not empty.

(4) The file storage units of the file shall be read or written only by formatted stream access input/output statements.

(5) Each file storage unit in the file is uniquely identified by a positive integer called the position. The first file storage unit in the file is at position 1. The relationship between positions of successive file storage units is processor dependent; not all positive integers need correspond to valid positions.

(6) The file position can be set to a position that was previously identified by the POS= specifier in INQUIRE.

NOTE 9.9

There may be some character positions in the file that do not correspond to characters written; this is because on some processors a record marker may be written to the file as a carriage-return/line-feed or other sequence. The means of determining the position in a file connected for stream acess is via the POS= specifier in an INQUIRE statement (9.8.1.19).

9.2.3 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 or file storage unit. The **terminal point** is the position just after the last record or file storage unit. If there are no records or file storage units in the file, the initial point and the terminal point are the same position.

If a record 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 $1 < i \leq n$ 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 $n \geq 1$ 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 $1 \leq i < n$ 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 $n \geq 1$ 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.

For a file connected for stream access, the file position is either between two file storage units, at the initial point of the file, at the terminal point of the file, or undefined.

9.2.3.1 Advancing and nonadvancing input/output

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

A **nonadvancing input/output statement** may position a record file at a character position within the current record, or a subsequent record (10.7.2). 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. If a nonadvancing output statement leaves a file positioned within a current record and no further output statement is executed for the file before it is closed or a BACKSPACE, ENDFILE, or REWIND statement is executed for it, the effect is as if the output statement were the corresponding advancing output statement.

9.2.3.2 File position prior to data transfer

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

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 REC= specifier. This record becomes the current record.

For stream access, the file is positioned immediately before the file storage unit specified by the POS= specifier; if there is no POS= specifier, the file position is not changed.

File positioning for child data transfer statements is described in 9.5.4.4.3.

9.2.3.3 File position after data transfer

If an error condition (9.5.3) occurred, the position of the file is indeterminate. If no error condition occurred, but an end-of-file condition (9.5.3) occurred as a result of reading an endfile record, the file is positioned after the endfile record.

For unformatted stream access, if no error condition occurred, the file position is not changed. For unformatted stream output, if the file position exceeds the previous terminal point of the file, the terminal point is set to the file position.

NOTE 9.10

An unformatted stream output statement with a POS= specifier and an empty output list can have the effect of extending the terminal point of a file without actually writing any data.

For a formatted stream output statement, if no error condition occurred, the terminal point of the file is set to the highest-numbered position to which data was transfered by the statement.

NOTE 9.11

The highest-numbered position might not be the current one if the output involved T or TL edit descriptors (10.7.1.1).

For formatted stream input, if an end-of-file condition occurred, the file position is not changed.

For nonadvancing input, if no error condition or end-of-file condition occurred, but an end-of-record condition (9.5.3) 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.

9.2.4 File storage units

A **file storage unit** is the basic unit of storage in a stream file or an unformatted record file. It is the unit of file position for stream access, the unit of record length for unformatted files, and the unit of file size for all external files.

Every value in a stream file or an unformatted record file shall occupy an integer number of file storage units; this number shall be the same for all scalar values of the same type and type parameters. The number of file storage units required for an item of a given type and type parameters may be determined using the IOLENGTH= specifier of the INQUIRE statement (9.8.3).

For a file connected for unformatted stream access, the processor shall not have alignment restrictions that prevent a value of any type from being stored at any positive integer file position.

It is recommended that the file storage unit be an 8-bit octet where this choice is practical.

NOTE 9.12

The requirement that every data value occupy an integer number of file storage units implies that data items inherently smaller than a file storage unit will require padding. This suggests that the file storage unit be small to avoid wasted space. Ideally, the file storage unit would be choosen such that padding is never required. A file storage unit of one bit would always meet this goal, but would likely be impractical because of the alignment requirements.

The prohibition on alignment restrictions prohibits the processor from requiring data alignments larger than the file storage unit.

The 8-bit octet is recommended as a good compromise that is small enough to accomodate the requirements of many applications, yet not so small that the data alignment requirements are likely to cause significant performance problems.

9.3 Internal files

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

An internal file is a record file with 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, except for child data transfer statements(9.5.4.4.3). 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 (9.5.4.4.2).

(9) On list-directed output, character constants are not delimited (10.9.2).

(10) The initial I/O rounding and sign modes are processor dependent.

(11) The initial decimal edit mode is POINT.

(12) Reading and writing records shall be accomplished only by sequential access formatted input/output statements.

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

9.4 File connection

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

J3 internal note

Unresolved isue 339

The bnf term external-file-unit (R902) is now inappropriate because it can sometimes be used for files that are not external. Notably in child data transfer statements, where it might be connected to an internal unit. None of the occurances actually depend on this terminology. Several statements do use the bnf term in their bnf in places where an internal unit won't do. But none of them rely on the bnf for that restriction. The only way this syntax could refer to an internal unit is in a child data transfer statement and we prohibit all the questionable statements on a unit while a parent is active except inquire, which we handle explicitly). The defining thing about this bnf term is not that it represents an external file, but that it is a number.

I suggest a global change of <external-file-unit> to <file-unit-number>, with a few corresponding "an" -> "a" changes where applicable. Is one case in c12 and several in c09. None elsewhere. Slight rewording of the first few paras of 9.4.0 also appropriate.

Internal-file-unit is probably ok (and appears only one place in the document other than the 3 occurances in 9.4.0; that one place is in 16.7.7(6).

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*

C901 (R902) 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 the value of a scalar integer expression that identifies an external file unit is negative, it shall either be equal to a **unit** argument of a currently active derived-type input/output procedure or equal to one of the named constants INPUT_UNIT, OUTPUT_UNIT, or ERROR_UNIT of the ISO_FORTRAN_ENV intrinsic module (13.12).

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 9.13

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 (9.5.4.2). These units are also identified by unit numbers defined by the named constants INPUT_UNIT and OUTPUT_UNIT of the ISO_FORTRAN_ENV intrinsic module (13.12).

This standard identifies a processor-dependent external unit for the purpose of error reporting. This unit shall be preconnected for sequential formatted output. The processor may define this to be the same as the output unit identified by an asterisk. This unit is also identified by a unit number defined by the named constant ERROR_UNIT of the ISO_FORTRAN_ENV intrinsic module.

9.4.1 Connection modes

A connection for formatted input/output has several changeable modes: the blank interpretation mode (10.7.6), delimiter mode (10.9.2, 10.10.2.1), sign mode (10.7.4), decimal edit mode (10.7.8), I/O rounding mode (10.7.7), pad mode (9.5.4.4.2), and scale factor (10.7.5). A connection for unformatted input/output has no changeable modes.

Values for the modes of a connection are established when the connection is initiated. If the connection is initiated by an OPEN statement, the values are as specified, either explicitly or implicitly, by the OPEN statement. If the connection is initiated other than by an OPEN statement (that is, if the file is an internal file or preconnected file) the values established are those that would be implied by an initial OPEN statement without the corresponding keywords.

The scale factor cannot be explicitly specified in an OPEN statement; it is implicitly 0.

The modes of a connection to an external file may be permanently changed by a subsequent OPEN statement that modifies the connection.

The modes of a connection may be temporarily changed by a corresponding keyword specifier in a data transfer statement or by an edit descriptor. Keyword specifiers take effect at the beginning of execution of the data transfer statement. Edit descriptors take effect when they are encountered in format processing. When a data transfer statement terminates, the values for the modes are reset to the values in effect immediately before the data transfer statement was executed.

9.4.2 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 CLOSE, INQUIRE, and WAIT statements also may refer to units that do not exist.

9.4.3 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.

Every input/output statement except an OPEN, CLOSE, INQUIRE, or WAIT 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).

NOTE 9.14

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.

This standard defines means of portable interoperation with C. C streams are described in 7.19.2 of the C standard. Whether a unit may be connected to a file which is also connected to a C stream is

processor dependent. It is processor dependent whether the files connected to the units INPUT_UNIT, OUTPUT_UNIT, and ERROR_UNIT correspond to the predefined C text streams standard input, standard output, and standard error.

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 9.15

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 that identifies the unit.

NOTE 9.16

For more explanatory information on file connection properties, see C.6.5.

9.4.4 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.

9.4.5 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 modes 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 modes 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 specifiers for changeable modes (9.4.1) 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 the value OLD or SCRATCH; the value SCRATCH is permitted only if the file to which the unit is already connected has a status of SCRATCH. Execution of such an OPEN statement causes any new values of the specifiers for changeable modes 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=, IOSTAT=, and IOMSG= 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.

If an existing file is not connected, execution of an OPEN statement that connects that file with a STATUS of SCRATCH is not permitted.

R904 *open-stmt* **is** OPEN (*connect-spec-list*)

R905 *connect-spec* **is** [UNIT =] *external-file-unit*
or ACCESS = *scalar-default-char-expr*
or ACTION = *scalar-default-char-expr*
or ASYNCHRONOUS = *scalar-default-char-expr*
or BLANK = *scalar-default-char-expr*
or DECIMAL = *scalar-default-char-expr*
or DELIM = *scalar-default-char-expr*
or ERR = *label*
or FILE = *file-name-expr*
or FORM = *scalar-default-char-expr*
or IOMSG = *iomsg-variable*
or IOSTAT = *scalar-int-variable*
or PAD = *scalar-default-char-expr*
or POSITION = *scalar-default-char-expr*
or RECL = *scalar-int-expr*
or ROUND = *scalar-default-char-expr*
or SIGN = *scalar-default-char-expr*
or STATUS = *scalar-default-char-expr*

R906 *file-name-expr* **is** *scalar-default-char-expr*

R907 *iomsg-variable* **is** *scalar-default-char-variable*

C902 (R905) No specifier shall appear more than once in a given *connect-spec-list*.

C903 (R905) An *external-file-unit* shall be specified; if the optional characters UNIT= are omitted from the unit specifier, the unit specifier shall be the first item in the *connect-spec-list*.

C904 (R905) 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 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. The value specified is without regard to case. Some specifiers have a default value if the specifier is omitted.

The IOSTAT=, ERR=, and IOMSG= specifiers are described in 9.9.

NOTE 9.17

An example of an OPEN statement is:

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

NOTE 9.18

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

9.4.5.1 ACCESS= specifier in the OPEN statement

The *scalar-default-char-expr* shall evaluate to SEQUENTIAL, DIRECT, or STREAM. The ACCESS= specifier specifies the access method for the connection of the file as being sequential, direct, or stream. 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.

9.4.5.2 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 input/output 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.

9.4.5.3 ASYNCHRONOUS= specifier in the OPEN statement

The *scalar-default-char-expr* shall evaluate to YES or NO. If YES is specified, asynchronous input/output on the unit is allowed. If NO is specified, asynchronous input/output on the unit is not allowed. If this specifier is omitted, the default value is NO.

9.4.5.4 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 connection for formatted input/output. It specifies the current value of the blank interpretation mode (10.7.6, 9.5.1.5) for input for this connection. This mode has no effect on output. It is a changeable mode (9.4.1). If this specifier is omitted in an OPEN statement that initiates a connection, the default value is NULL.

9.4.5.5 DECIMAL= specifier in the OPEN statement

The *scalar-default-char-expr* shall evaluate to COMMA or POINT. The DECIMAL= specifier is permitted only for a connection for formatted input/output. It specifies the current value of the decimal edit mode (10.7.8, 9.5.1.6) for this connection. This is a changeable mode (9.4.1). If this specifier is omitted in an OPEN statement that initiates a connection, the default value is POINT.

9.4.5.6 DELIM= specifier in the OPEN statement

The *scalar-default-char-expr* shall evaluate to APOSTROPHE, QUOTE, or NONE. The DELIM= specifier is permitted only for a connection for formatted input/output. It specifies the current value of the delimiter mode (9.5.1.7) for list-directed (10.9.2) and namelist (10.10.2.1) output for the connection. This mode has no effect on input. It is a changeable mode (9.4.1). If this specifier is omitted in an OPEN statement that initiates a connection, the default value is NONE.

9.4.5.7 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. The interpretation of case is processor dependent.

9.4.5.8 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 or stream 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.

9.4.5.9 PAD= specifier in the OPEN statement

The *scalar-default-char-expr* shall evaluate to YES or NO. The PAD= specifier is permitted only for a connection for formatted input/output. It specifies the current value of the pad mode (9.5.4.4.2, 9.5.1.9) for input for this connection. This mode has no effect on output. It is a changeable mode (9.4.1). If this specifier is omitted in an OPEN statement that initiates a connection, the default value is YES.

NOTE 9.19

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

9.4.5.10 POSITION= specifier in the OPEN statement

The *scalar-default-char-expr* shall evaluate to ASIS, REWIND, or APPEND. The connection shall be for sequential or stream 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.

9.4.5.11 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 not be present when a file is being connected for stream 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 file storage 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.

9.4.5.12 ROUND= specifier in the OPEN statement

The *scalar-default-char-expr* shall evaluate to one of UP, DOWN, ZERO, NEAREST, COMPATIBLE, or PROCESSOR_DEFINED. The ROUND= specifier is permitted only for a connection for formatted input/output. It specifies the current value of the I/O rounding mode (10.7.7, 9.5.1.12) for this connection. This is a changeable mode (9.4.1). If this specifier is omitted in an OPEN statement that initiates a connection, the I/O rounding mode is processor dependent; it shall be one of the above modes.

NOTE 9.20

A processor is free to select any I/O rounding mode for the default mode. The mode might correspond to UP, DOWN, ZERO, NEAREST, or COMPATIBLE; or it might be a completely different I/O rounding mode.

9.4.5.13 SIGN= specifier in the OPEN statement

The *scalar-default-char-expr* shall evaluate to one of PLUS, SUPPRESS, or PROCESSOR_DEFINED. The SIGN= specifier is permitted only for a connection for formatted input/output. It specifies the current value of the sign mode (10.7.4, 9.5.1.13) for this connection. This is a changeable mode (9.4.1). If this specifier is omitted in an OPEN statement that initiates a connection, the default value is PROCESSOR_DEFINED.

9.4.5.14 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 normal termination of the program.

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

9.4.6 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 for 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 performs a wait operation for any pending asynchronous data transfer operations for the specified 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 normal termination of execution of a program, 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 9.21

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

R908 *close-stmt* **is** CLOSE (*close-spec-list*)

R909 *close-spec* **is** [UNIT =] *external-file-unit*
or IOSTAT = *scalar-int-variable*

or IOMSG = *iomsg-variable*
or ERR = *label*
or STATUS = *scalar-default-char-expr*

C905 (R909) No specifier shall appear more than once in a given *close-spec-list*.

C906 (R909) An *external-file-unit* shall be specified; if the optional characters UNIT= are omitted from the unit specifier, the unit specifier shall be the first item in the *close-spec-list*.

C907 (R909) 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. The value specified is without regard to case.

The IOSTAT=, ERR=, and IOMSG= specifiers are described in 9.9.

NOTE 9.22

An example of a CLOSE statement is:

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

NOTE 9.23

For more explanatory information on the CLOSE statement, see C.6.6.

9.4.6.1 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.

9.5 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.

R910 *read-stmt* **is** READ (*io-control-spec-list*) [*input-item-list*]
or READ *format* [, *input-item-list*]

R911 *write-stmt* **is** WRITE (*io-control-spec-list*) [*output-item-list*]

R912 *print-stmt* **is** PRINT *format* [, *output-item-list*]

NOTE 9.24

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)
```

9.5.1 Control information list

A **control information list** is an *io-control-spec-list*. It governs data transfer.

R913 *io-control-spec* **is** [UNIT =] *io-unit*

or [FMT =] *format*
or [NML =] *namelist-group-name*
or ADVANCE = *scalar-default-char-expr*
or ASYNCHRONOUS = *scalar-char-initialization-expr*
or BLANK = *scalar-default-char-expr*
or DECIMAL = *scalar-default-char-expr*
or DELIM = *scalar-default-char-expr*
or END = *label*
or EOR = *label*
or ERR = *label*
or ID = *scalar-int-variable*
or IOMSG = *iomsg-variable*
or IOSTAT = *scalar-int-variable*
or PAD = *scalar-default-char-expr*
or POS = *scalar-int-expr*
or REC = *scalar-int-expr*
or ROUND = *scalar-default-char-expr*
or SIGN = *scalar-default-char-expr*
or SIZE = *scalar-int-variable*

C908 (R913) No specifier shall appear more than once in a given *io-control-spec-list*.

C909 (R913) An *io-unit* shall be specified; if the optional characters UNIT= are omitted from the unit specifier, the unit specifier shall be the first item in the *io-control-spec-list*.

C910 (R913) A DELIM= or SIGN= specifier shall not appear in a *read-stmt*.

C911 (R913) A BLANK=, PAD=, END=, EOR=, or SIZE= specifier shall not appear in a *write-stmt*.

C912 (R913) 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.

C913 (R913) A *namelist-group-name* shall be the name of a namelist group.

C914 (R913) 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.

C915 (R913) An *io-control-spec-list* shall not contain both a *format* and a *namelist-group-name*.

C916 (R913) 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=.

C917 (R913) 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=.

C918 (R913) If the unit specifier specifies an internal file, the *io-control-spec-list* shall not contain a REC= specifier or a POS= specifier.

C919 (R913) 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.

C920 (R913) An ADVANCE= specifier may be present only in a formatted sequential or stream input/output statement with explicit format specification (10.1) whose control information list does not contain an internal file unit specifier.

C921 (R913) If an EOR= specifier is present, an ADVANCE= specifier also shall appear.

C922 (R913) If a SIZE= specifier is present, an ADVANCE= specifier also shall appear.

C923 (R913) The *scalar-char-initialization-expr* in an ASYNCHRONOUS= specifier shall be of type default character and shall have the value YES or NO.

C924 (R913) An ASYNCHRONOUS= specifier with a value YES shall not appear unless *io-unit* is an *external-file-unit*.

C925 (R913) If an ID= specifier is present, an ASYNCHRONOUS= specifier with the value YES shall also appear.

C926 (R913) If a POS= specifier is present, the *io-control-spec-list* shall not contain a REC= specifier.

C927 (R913) If a DECIMAL=, BLANK=, PAD=, SIGN=, or ROUND= specifier is present, a *format* or *namelist-group-name* shall also appear.

C928 (R913) If a DELIM= specifier is present, either *format* shall be an asterisk or *namelist-group-name* 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**.

The ADVANCE=, ASYNCHRONOUS=, DECIMAL=, BLANK=, DELIM=, PAD=, SIGN=, and ROUND= specifiers have a limited list of character values. Any trailing blanks are ignored. The values specified are without regard to case.

The IOSTAT=, ERR=, EOR=, END=, and IOMSG= specifiers are described in 9.9.

NOTE 9.25

An example of a READ statement is:

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

9.5.1.1 FMT= specifier in a data transfer statement

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

R914 *format* **is** *default-char-expr*
or *label*
or *

C929 (R914) 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 9.26

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 9.27

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.

9.5.1.2 NML= specifier in a data transfer statement

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**.

9.5.1.3 ADVANCE= specifier in a data transfer statement

The *scalar-default-char-expr* shall evaluate to YES or NO. The ADVANCE= specifier determines whether advancing input/output occurs for this input/output statement. If YES is specified, advancing input/output occurs. If NO is specified, nonadvancing input/output occurs (9.2.3.1). If this specifier is omitted from an input/output statement that allows the specifier, the default value is YES.

9.5.1.4 ASYNCHRONOUS= specifier in a data transfer statement

The ASYNCHRONOUS= specifier determines whether this input/output statement is synchronous or asynchronous. If YES is specified, the statement and the input/output operation are said to be asynchronous. If NO is specified or if the specifier is omitted, the statement and the input/output operation are said to be synchronous.

Asynchronous input/output is permitted only for external files opened with an ASYNCHRONOUS= specifier with the value YES in the OPEN statement.

NOTE 9.28

Both synchronous and asynchronous input/output are allowed for files opened with an ASYNCHRONOUS= specifier of YES. For other files, only synchronous input/output is allowed; this includes files opened with an ASYNCHRONOUS= specifier of NO, files opened without an ASYNCHRONOUS= specifier, preconnected files accessed without an OPEN statement, and internal files.

The ASYNCHRONOUS= specifier value in a data transfer statement is an initialization expression because it effects compiler optimizations and, therefore, needs to be known at compile time.

The processor may perform an asynchronous data transfer operation asynchronously, but it is not required to do so. Records and file storage units read or written by asynchronous data transfer statements are read, written, and processed in the same order as they would have been if the data transfer statements were synchronous.

If a variable is used in an asynchronous data transfer statement as

(1) an item in an input/output list,

(2) a group object in a namelist, or

(3) a SIZE= specifier

the base object of the *data-ref* is implicitly given the ASYNCHRONOUS attribute in the scoping unit of the data transfer statement. This attribute may be confirmed by explicit declaration.

When an asynchronous input/output statement is executed, the set of storage units specified by the item list or namelist specifier, plus the storage units specified by the SIZE= specifier, is defined to be the pending input/output storage sequence for the data transfer operation.

NOTE 9.29

A pending input/output storage sequence is not necessarily a contiguous set of storage units.

Within a scoping unit, a pending input/output storage sequence **affector** is a variable that is referenced or defined, and whose definition or reference refers to any storage unit in any pending

input/output storage sequence. A variable may be a pending input/output storage sequence affector in one scoping unit but not in some other scoping unit because pending input/output storage sequences exist only for the duration of an asynchronous data transfer operation (9.6).

9.5.1.5 BLANK= specifier in a data transfer statement

The *scalar-default-char-expr* shall evaluate to NULL or ZERO. The BLANK= specifier temporarily changes (9.4.1) the blank interpretation mode (10.7.6, 9.4.5.4) for the connection. If the specifier is omitted, the mode is not changed.

9.5.1.6 DECIMAL= specifier in a data transfer statement

The *scalar-default-char-expr* shall evaluate to COMMA or POINT. The DECIMAL= specifier temporarily changes (9.4.1) the decimal edit mode (10.7.8, 9.4.5.5) for the connection. If the specifier is omitted, the mode is not changed.

9.5.1.7 DELIM= specifier in a data transfer statement

The *scalar-default-char-expr* shall evaluate to APOSTROPHE, QUOTE, or NONE. The DELIM= specifier temporarily changes (9.4.1) the delimiter mode (10.9.2, 10.10.2.1, 9.4.5.6) for the connection. If the specifier is omitted, the mode is not changed.

9.5.1.8 ID= specifier in a data transfer statement

Sucessful execution of an asynchronous data transfer statement containing an ID= specifier causes the variable specified in the ID= specifier to become defined with a processor-dependent value. This value is referred to as the identifier of the data transfer operation. It can be used in a subsequent WAIT or INQUIRE statement to identify the particular data transfer operation.

If an error occurs during the execution of a data transfer statement containing an ID= specifier, the variable specified in the ID= specifier becomes undefined.

A child data transfer statement shall not specify the ID= specifier.

9.5.1.9 PAD= specifier in a data transfer statement

The *scalar-default-char-expr* shall evaluate to YES or NO. The PAD= specifier temporarily changes (9.4.1) the pad mode (9.5.4.4.2, 9.4.5.9) for the connection. If the specifier is omitted, the mode is not changed.

9.5.1.10 POS= specifier in a data transfer statement

The POS= specifier specifies the file position in file storage units. This specifier may be present only in an input/output statement that specifies a unit connected for stream access. A child data transfer statement shall not specify this specifier.

A processor may prohibit the use of POS= with particular files that do not have the properties necessary to support random positioning. A processor may also prohibit positioning a particular file to any position prior to its current file position if the file does not have the properties necessary to support such positioning.

NOTE 9.30

A file that represents connection to a device or data stream might not be positionable.

If the file is connected for formatted stream access, the file position specified by POS= shall be equal to either 1 (the beginning of the file) or a value previously returned by a POS= specifier in an INQUIRE statement for the file.

9.5.1.11 REC= specifier in a data transfer statement

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; it shall not be specified in a child data transfer statement. If the control information list contains a REC= specifier, the statement is a **direct access input/output statement**. A child data transfer statement is a direct access data transfer statement if the parent is a direct access data transfer statement. Any other data transfer statement is a **sequential access input/output statement** or a **stream access input/output statement**, depending on whether the file connection is for sequential access or stream access.

9.5.1.12 ROUND= specifier in a data transfer statement

The *scalar-default-char-expr* shall evaluate to one of the values specified in 9.4.5.12. The ROUND= specifier temporarily changes (9.4.1) the I/O rounding mode (10.7.7, 9.4.5.12) for the connection. If the specifier is omitted, the mode is not changed.

9.5.1.13 SIGN= specifier in a data transfer statement

The *scalar-default-char-expr* shall evaluate to PLUS, SUPPRESS, or PROCESSOR_DEFINED. The SIGN= specifier temporarily changes (9.4.1) the sign mode (10.7.4, 9.4.5.13) for the connection. If the specifier is omitted, the mode is not changed.

9.5.1.14 SIZE= specifier in a data transfer statement

When a synchronous 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 (9.5.4.4.2) are not counted.

For asynchronous nonadvancing input, the storage units specified in the SIZE= specifier become defined with the count of the characters transferred when the corresponding wait operation is executed.

9.5.2 Data transfer input/output list

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

R915 *input-item* **is** *variable*
or *io-implied-do*

R916 *output-item* **is** *expr*
or *io-implied-do*

R917 *io-implied-do* **is** (*io-implied-do-object-list* , *io-implied-do-control*)

R918 *io-implied-do-object* **is** *input-item*
or *output-item*

R919 *io-implied-do-control* **is** *do-variable* = *scalar-int-expr* , ■
■ *scalar-int-expr* [, *scalar-int-expr*]

C930 (R915) A variable that is an *input-item* shall not be a whole assumed-size array.

C931 (R915) A variable that is an *input-item* shall not be a procedure pointer.

C932 (R919) The *do-variable* shall be a named scalar variable of type integer.

C933 (R918) 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*.

C934 (R916) An expression that is an *output-item* shall not have a value that is a procedure pointer.

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 9.31

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).

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

A list item shall not be polymorphic unless it is processed by a user-defined derived-type input/output procedure (9.5.4.4.3).

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*.

The rules describing whether to expand an input/output list item are re-applied to each expanded list item until none of the rules apply.

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 9.32

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
```

If a list item of derived type in an unformatted input/output statement is not processed by a user-defined derived-type input/output procedure (9.5.4.4.3), and if any subobject of that list item would be processed by a user-defined derived-type input/output procedure, the list item is treated as if all of the components of the object were specified in the list in component order (4.5.4); those components shall be accessible in the scoping unit containing the input/output statement and shall not be pointers or allocatables.

If a list item of derived type in a formatted input/output statement is not processed by a user-defined derived-type input/output procedure, that list item is treated as if all of the components of the list item were specified in the list in component order; those components shall be accessible in the scoping unit containing the input/output statement and shall not be pointers or allocatables.

If a derived-type list item is not treated as a list of its individual components, that list item's ultimate components shall not be pointers or allocatables unless that list item is processed by a user-defined derived-type input/output procedure.

The scalar objects resulting when a data transfer statement's list items are expanded according to the rules in this section for handling array and derived-type list items are called **effective items**.

Zero-sized arrays and implied-DO lists with an iteration count of zero do not contribute to the effective list items. A scalar character item of zero length is an effective list item.

NOTE 9.33

In a formatted input/output statement, edit descriptors are associated with effective list items, which are always scalar. The rules in 9.5.2 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.

An effective input/output list item of derived type in an unformatted input/output statement is treated as a single value in a processor-dependent form unless the list item is processed by a user-defined derived-type input/output procedure (9.5.4.4.3).

NOTE 9.34

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 (8.1.5.4).

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

NOTE 9.35

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 9.36

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
```

9.5.3 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, unless the file is a stream file and the current record at the end of the file (an end-of-file condition occurs instead).

An **end-of-file condition** occurs in 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.

(3) When an attempt is made to read beyond the end of a stream 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. An end-of-file condition also may occur during execution of a stream input statement.

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 or namelist group objects become undefined.

If an end-of-record condition occurs during execution of a nonadvancing input statement, the following occurs: if the pad mode has the value YES, the record is padded with blanks (9.5.4.4.2) 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.

If an error condition occurs during execution of an input/output statement that contains an IOSTAT= or ERR=specifier, an end-of-file condition occurs during execution of an input/output statement that contains an IOSTAT= or END= specifier, or and end-of-record condition occurs during execution of an input/output statement that contains an IOSTAT= or EOR= specifier, then execution continues as specified in 9.9.

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

9.5.4 Execution of a data transfer input/output statement

Execution of a WRITE or PRINT statement for a file that does not exist creates the file unless an error condition occurs.

The effect of executing a synchronous 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) Perform a wait operation for all pending input/output operations for the unit. If an error, end-of-file, or end-of-record condition occurs during any of the wait operations, steps 4 through 8 are skipped for the current data transfer statement.

(4) Establish the format if one is specified.

(5) Position the file prior to data transfer (9.2.3.2) unless the statement is a child data transfer statement (9.5.4.4.3).

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

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

(8) Position the file after data transfer (9.2.3.3) unless the statement is a child data transfer statement (9.5.4.4.3).

(9) Cause any variable specified in a SIZE= specifier to become defined.

(10) If an error, end-of-file, or end-of-record condition (9.5.3) occured, processing continues as specified in 9.9; otherwise any variable specified in an IOSTAT= specifier is assigned the value zero.

The effect of executing an asynchronous 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 (9.2.3.2).

(5) Establish the set of storage units identified by the input/output list. For a READ statement, this might require some or all of the data in the file to be read if an input variable is used in an implied-DO in the input/output list, as a *subscript*, *substring-range*, *stride*, or is otherwise referenced.

(6) Initiate an asynchronous data transfer between the file and the entities specified by the input/output list (if any) or namelist. The asynchronous data transfer may complete (and an error, end-of-file, or end-of-record condition may occur) during the execution of this data transfer statement or during a later wait operation.

(7) Determine whether an error, end-of-file, or end-of-record condition has occurred. The conditions may occur during the execution of this data transfer statement or during the corresponding wait operation, but not both.

Also, any of these conditions that would have occured during the corresponding wait operation for a previously pending data transfer operation that does not have an identifier may occur during the execution of this data transfer statement.

(8) Position the file as if the data transfer had finished (9.2.3.3).

(9) Cause any variable specified in a SIZE= specifier to become undefined.

(10) If an error, end-of-file, or end-of-record condition (9.5.3) occured, processing continues as specified in 9.9; otherwise any variable specified in an IOSTAT= specifier is assigned the value zero.

For an asynchronous data transfer statement, the data transfers may occur during execution of the statement, during execution of the corresponding wait operation, or anywhere between. The data transfer operation is considered to be pending until a corresponding wait operation is performed.

For asynchronous output, a pending input/output storage sequence affector (9.5.1.4) shall not be redefined, become undefined, or have its accessibility or association status changed.

For asynchronous input, a pending input/output storage sequence affector shall not be referenced, become defined, become undefined, become associated with a dummy argument that has the VALUE attribute, or have its accessibility or association status changed.

Error, end-of-file, and end-of-record conditions in an asynchronous data transfer operation may occur during execution of either the data transfer statement or the corresponding wait operation. If an ID= specifier is not present in the initiating data transfer statement, the conditions may occur during the execution of any subsequent data transfer or wait operation for the same unit. When a condition occurs for a previously executed asynchronous data transfer statement, a wait operation is performed for all pending data transfer operations on that unit. When a condition occurs during a subsequent statement, any actions specified by IOSTAT=, IOMSG=, ERR=, END=, and EOR= specifiers for that statement are taken.

NOTE 9.37

Because end-of-file and error conditions for asynchronous data transfer statements without an ID= specifier may be reported by the processor during the execution of a subsequent data transfer statement, it may be impossible for the user to determine which input/output statement caused the condition. Reliably detecting which READ statement caused an end-of-file condition requires that all asynchronous READ statements for the unit include an ID= specifier.

9.5.4.1 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.

9.5.4.2 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 9.38

The file may be preconnected.

9.5.4.3 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.

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.

9.5.4.4 Data transfer

Data are transferred between the file and the 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 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*. The next effective item to be processed is called the next effective item. Effective items are derived from the input/output list items as described in 9.5.2.

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 9.39

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 (10.10) 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 9.40

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.

9.5.4.4.1 Unformatted data transfer

During unformatted data transfer, data are transferred without editing between the file and the entities specified by the input/output list. If the file is connected for sequential or direct access, exactly one record is read or written.

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

The number of file storage units required by the input list shall be less than or equal to the number of file storage units in the record. A value in the file is stored in a contiguous sequence of file storage units, beginning with the file storage unit immediately following the current file position.

After each value is transferred, the current file position is moved to a point immediately after the last file storage unit of the value.

On input, if the file storage units transferred do not contain a value with the same type and type parameters as the input list entity, then the resulting value of the entity is processor-dependent except in the following cases:

(1) A complex list entity may correspond to two real values of the same kind stored in the file, or vice-versa.

(2) A default character list entity of length *n* may correspond to *n* default characters stored in the file, regardless of the length parameters of the entities that were written to these storage units of the file. If the file is connected for stream input, the characters may have been written by formatted stream output.

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 not connected for unformatted input/output, unformatted data transfer is prohibited.

The unit specified shall be an external unit.

9.5.4.4.2 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*. 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 unless the derived-type list item is processed by a user-defined derived-type input/output procedure (9.5.4.4.3).

If the file is not connected for formatted input/output, formatted data transfer is prohibited.

During advancing input when the pad mode 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 when the pad mode 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 when the pad mode has the value NO, an end-of-record condition (9.5.3) occurs if the input list and format specification require more characters from the record than the record contains, and the record is not incomplete (9.2.2.3). If the record is incomplete, an end-of-file condition occurs instead of an end-of-record condition.

During nonadvancing input when the pad mode has the value YES, 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 theh record is incomplete, an end-of-file condition occurs; otherwise and end-of-record condition occurs.

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.

9.5.4.4.3 User-defined derived-type input/output procedures

User-defined derived-type input/output procedures allow a program to alter the default handling of derived-type objects and values in data transfer input/output statements as described in 9.5.2.

J3 internal note

Unresolved issue 333

The material defining whether or not dtio happens for a particular effective io item is scattered all over the document in a way that makes it hard to find and understand. This is not entirely new to paper 01-250. Section 9.5.2 is the main place where the issue of expanding the i/o list to obtain effective items is discussed. This expansion depends, among other things, on whether dtio processing is done for an item. The section has several xrefs on that question, all pointing to 9.5.4.4.3. But 9.5.4.4.3 no longer has any material defining whether or not dtio processing happens. It xrefs 14.1.2.4.4, which is where the material really is.

A section on resolving procedure references (14.1.2.4.4) seems a strange place for the material about whether dtio is permitted (list item 1). Further, the wording of that list item (in particular, the "that is") implies that it is summarizing material from elsewhere. This appears to be the only place where this is actually said, making the "that is" inappropriate.

The 3rd para of 9.5.4.4.3 then contradicts itself in multiple ways. After citing 14.1.2.4.4, it refers to "other requirements" in 10.6.5 and 12.3.2. As best as I can see, this is irrelevant and misleading. If a dtio procedure is selected as described in 14.1.2.4.4, then dtio is done. Period. If there are other requirements that are not met, then the program is illegal. There are no valid cases in which a procedure is selected as described in 14, but is not used. Perhaps this para is a remnant of some former draft in which some of the material was not in 14.1.2.4.4.

Finally, the 3rd para of 9.5.4.4.3 says that if a dtiop is used, the list items are not processed as described in 9.5.2. However, 9.5.2 specifically mentions the exceptions for dtio. So this statement is self-contradictory, in essense saying that if dtio processing is done, then dtio processing is not done.

A user-defined derived-type input/output procedure is a procedure accessible by a *dtio-generic-spec* (12.3.2.1). A particular user-defined derived-type input/output procedure is selected as described in 16.1.2.4.4. If the selection results in a disassociated procedure pointer, a deferred binding, or a dummy procedure or dummy procedure pointer that is not present, an error condition occurs.

If a derived type input/output procedure is selected as specified in the previous paragraph, and other requirements are met (10.6.5, 12.3.2), the processor shall not process those derived-type list items as described in 9.5.2. Instead, it shall call the corresponding user-defined derived-type input/output procedure for any data transfer input/output statements executed in that scoping unit. The user-defined derived-type input/output procedure controls the actual data transfer operations for the derived-type list item.

A data transfer statement that includes a derived-type list item and that causes a user-defined derived-type input/output procedure to be invoked is called a **parent data transfer statement**. A data transfer statement that is executed while a parent data transfer statement is being processed and that specifies the unit passed into a user-defined derived-type input/output procedure is called a **child data transfer statement**.

NOTE 9.41

A user-defined derived-type input/output procedure will usually contain child data transfer statements that read values from or write values to the current record or at the current file position. The effect of executing the user-defined derived-type input/output procedure is similar to that of substituting the list items from any child data transfer statements into the parent data transfer statement's list items, along with similar substitutions in the format specification.

NOTE 9.42

A particular execution of a READ, WRITE or PRINT statement can be both a parent and a child data transfer statement. A user-defined derived-type input/output procedure can indirectly call itself or another user-defined derived-type input/output procedure by executing a child data transfer statement containing a list item of derived type, where a matching interface is accessible for that derived type. If a user-defined derived-type input/output procedure calls itself indirectly in this manner, it shall be declared RECURSIVE.

A child data transfer statement is processed differently from a nonchild data transfer statement in the following ways:

- Executing a child data transfer statement does not position the file prior to data transfer.
- An unformatted child data transfer statement does not position the file after data transfer is complete.

For a particular derived type and a particular set of kind type parameter values, there are four possible sets of characteristics for user-defined derived-type input/output procedures; one each for formatted input, formatted output, unformatted input, and unformatted output. The user need not supply all four procedures. The procedures are specified to be used for derived-type input/output by interface blocks (12.3.2.1) or by derived-type procedure bindings (4.5.1.5), with a *dtio-generic-spec*.

In the four interfaces, which specify the characteristics of user-defined procedures for derived-type input/output, the following syntax term is used:

R920 *dtv-type-spec* **is** TYPE(*derived-type-spec*)
or CLASS(*derived-type-spec*)

C935 (R920) If *derived-type-spec* specifies an extensible type, the CLASS keyword shall be used; otherwise, the TYPE keyword shall be used.

C936 (R920) All nonkind type parameters of *derived-type-spec* shall be assumed.

If the *generic-spec* is READ (FORMATTED) the characteristics shall be the same as those specified by the following interface:

```
SUBROUTINE my_read_routine_formatted                &
                            (unit,                  &
                             dtv,                   &
                             iotype, v_list,        &
                             eof, err, eor, iomsg)
  INTEGER, INTENT(IN) :: unit ! unit number
  ! the derived-type value/variable
  dtv-type-spec, INTENT(OUT) :: dtv
  ! the edit descriptor string
  CHARACTER (LEN=*), INTENT(IN) :: iotype
  INTEGER, INTENT(IN) :: v_list(:)
  LOGICAL, INTENT(OUT) :: eof, err, eor
  CHARACTER (LEN=*), INTENT(INOUT) :: iomsg
END
```

If the *generic-spec* is READ (UNFORMATTED) the characteristics shall be the same as those specified by the following interface:

```
SUBROUTINE my_read_routine_unformatted              &
                            (unit,                  &
                             dtv,                   &
                             eof, err, eor, iomsg)
  INTEGER, INTENT(IN) :: unit
  ! the derived-type value/variable
  dtv-type-spec, INTENT(OUT) :: dtv
```

```
  LOGICAL, INTENT(OUT) :: eof, err, eor
  CHARACTER (LEN=*), INTENT(INOUT) :: iomsg
END
```

If the *generic-spec* is WRITE (FORMATTED) the characteristics shall be the same as those specified by the following interface:

```
SUBROUTINE my_write_routine_formatted              &
                           (unit,                  &
                            dtv,                   &
                            iotype, v_list,        &
                            err, iomsg)
  INTEGER, INTENT(IN) :: unit
  ! the derived-type value/variable
  dtv-type-spec, INTENT(IN) :: dtv
  ! the edit descriptor string
  CHARACTER (LEN=*), INTENT(IN) :: iotype
  INTEGER, INTENT(IN) :: v_list(:)
  LOGICAL, INTENT(OUT) :: err
  CHARACTER (LEN=*), INTENT(INOUT) :: iomsg
END
```

If the *generic-spec* is WRITE (UNFORMATTED) the characteristics shall be the same as those specified by the following interface:

```
SUBROUTINE my_write_routine_unformatted            &
                           (unit,                  &
                            dtv,                   &
                            err, iomsg)
  INTEGER, INTENT(IN) :: unit
  ! the derived-type value/variable
  dtv-type-spec, INTENT(IN) :: dtv
  LOGICAL, INTENT(OUT) :: err
  CHARACTER (LEN=*), INTENT(INOUT) :: iomsg
END
```

The actual specific procedure names (the **my_..._routine_...** procedure names above) are not significant. In the discussion here and elsewhere, the dummy arguments in these interfaces are referred by the names given above; the names are, however, arbitrary.

In addition to the characteristics specified by the above interfaces, the **dtv** dummy argument may optionally have the VOLATILE attribute.

When a user-defined derived-type input/output procedure is invoked, the processor shall pass a **unit** argument that has a value as follows:

- If the parent data transfer statement uses an external unit other than an asterisk, the value of the **unit** argument shall be that of the external unit.
- If the parent data transfer statement is a WRITE statement with an asterisk unit or a PRINT statement, the **unit** argument shall have the same value as the OUTPUT_UNIT named constant of the ISO_FORTRAN_ENV intrinsic module (13.12).
- If the parent data transfer statement is a READ statement with an asterisk unit or a READ statement without an *io-control-spec-list*, the **unit** argument shall have the same value as the INPUT_UNIT named constant of the ISO_FORTRAN_ENV intrinsic module.
- Otherwise the parent data transfer statement must access an internal file, in which case the **unit** argument shall have a processor-dependent negative value.

NOTE 9.43

Because the unit argument value will be negative when the parent data transfer statement specifies an internal file, a user-defined derived-type input/output procedure should not execute an INQUIRE statement without checking that the **unit** argument is nonnegative or is equal to one of the named constants INPUT_UNIT, OUTPUT_UNIT, or ERROR_UNIT of the ISO_FORTRAN_ENV intrinsic module(13.12.1).

The processor shall pass an **iotype** argument that has a value as follows:

- "LISTDIRECTED" if the parent data transfer statement specified list directed formatting,
- "NAMELIST" if the parent data transfer statement specified namelist formatting, or
- "DT" concatenated with the *char-literal-constant*, if any, of the edit descriptor, if the parent data transfer statement contained a format specification and the list item's corresponding edit descriptor was a DT edit descriptor.

If the parent data transfer statement is a READ statement, the processor shall define the effective list item from the parent READ statement with the value assigned to the **dtv** dummy argument (or some portion thereof) by the user-defined derived-type input procedure.

If the parent data transfer statement is a WRITE or PRINT statement, the processor shall provide the value of the effective list item in the **dtv** dummy argument.

If the *v-list* of the edit descriptor is present in the parent data transfer statement, the processor shall provide the values from it in the **v_list** dummy argument, with the same number of elements in the same order as *v-list*. If there is no *v-list* in the edit descritor or if the data transfer statement specifies list-directed or namelist formatting, the processor shall provide **v_list** as a zero-sized array.

The **err**, **eof**, and **eor** arguments correspond, respectively, to the error, end-of-file, and end-of-record conditions (9.5.3). The user-defined derived-type input/output procedures for reads shall return values of true or false for these arguments; the user-defined derived-type input/output procedures for writes shall return values of true or false for the **err** argument. If one of these arguments has the value true when the user-defined derived-type input/output procedure returns, the corresponding condition is triggered by the processor in the parent data transfer statement. No more than one of these arguments shall have a true value when the user-defined derived-type input/output procedure returns.

If the derived-type input/output procedure returns the value .TRUE. for the **err**, **eof**, or **eor** argument, the procedure shall also return an explanatory message in the **iomsg** argument. Otherwise, the procedure shall not change the value of the **iomsg** argument.

If the **err**, **eof**, or **eor** argument of a user-defined derived-type input/output procedure has the value true when that procedure returns, and the processor therefore terminates execution of the program as described in 9.5.3, the processor shall make the value of the **iomsg** argument available. in a processor-dependent manner

NOTE 9.44

The user's procedure may chose to interpret an element of the **v_list** argument as a field width, but this is not required. If it does, it would be appropriate to fill an output field with "*"s if the width is too small.

When a parent READ statement is active, an input/output statement shall not read from any external unit other than the one specified by the dummy argument **unit** and shall not perform output to any external unit.

When a parent WRITE or PRINT statement is active, an input/output statement shall not perform output to any external unit other than the one specified by the dummy argument **unit** and shall not read from any external unit.

When a parent data transfer statement is active, a data transfer statement that specifies an internal file is permitted.

OPEN, CLOSE, BACKSPACE, ENDFILE, and REWIND statements shall not be executed while a parent data transfer statement is active.

A user-defined derived-type input/output procedure may use a FORMAT with a DT edit descriptor for handling a component of the derived type that is itself of a derived type. A child data transfer statement that is a list directed or namelist input/output statement may contain a list item of derived type.

Because a child data transfer statement does not position the file prior to data transfer, the child data transfer statement starts transferring data from where the file was positioned by the parent data transfer statement's most recently processed effective list item or record positioning edit descriptor. This is not necessarily at the beginning of a record.

A record positioning edit descriptor, such as TL and TR, used on **unit** by a child data transfer statement shall not cause the record position to be positioned before the record position at the time the user-defined derived-type input/output procedure was invoked.

NOTE 9.45

A robust user-defined derived-type input/output procedure may wish to use INQUIRE to determine the settings of BLANK=, PAD=, ROUND=, DECIMAL=, and DELIM= for an external unit. The INQUIRE provides values as specified in 9.8.

Neither a parent nor child data transfer statement shall be asynchronous.

A user-defined derived-type input/output procedure, and any procedures invoked therefrom, shall not define, nor cause to become undefined, any storage location referenced by any input/output list item, the corresponding format, or any specifer in any active parent data transfer statement, except through the **dtv** argument.

NOTE 9.46

A child data transfer statement shall not specify the ID=, POS=, or REC= specifiers in an input/output control list.

9.5.4.5 List-directed formatting

If list-directed formatting has been established, editing is performed as described in 10.9.

9.5.4.6 Namelist formatting

If namelist formatting has been established, editing is performed as described in 10.10.

Every allocatable *namelist-group-object* in the namelist group shall be allocated and every *namelist-group-object* that is a pointer shall be associated with a target. If a namelist-group-object is polymorphic or has an ultimate component that is allocatable or a pointer, that object shall be processed by a user-defined derived-type input/output procedure (9.5.4.4.3).

9.5.5 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 (10.9, 10.10) 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 (9.5.3).

9.6 Waiting on pending data transfer

Execution of an asynchronous data transfer statement in which neither an error, end-of-record, nor end-of-file condition occurs initiates a pending data transfer operation. There may be multiple pending data transfer operations for the same or multiple units simultaneously. A pending data transfer operation remains pending until a corresponding wait operation is performed. A wait operation may be performed by a WAIT, INQUIRE, CLOSE, or file positioning statement.

9.6.1 WAIT statement

A WAIT statement performs a wait operation for specified pending asynchronous data transfer operations.

NOTE 9.47

The CLOSE, INQUIRE, and file positioning statements may also perform wait operations.

R921 *wait-stmt* **is** WAIT (*wait-spec-list*)

R922 *wait-spec* **is** [UNIT =] *external-file-unit*
or END = *label*
or EOR = *label*
or ERR = *label*
or ID = *scalar-int-variable*
or IOMSG = *iomsg-variable*
or IOSTAT = *scalar-int-variable*

C937 (R922) No specifier shall appear more than once in a given *wait-spec-list*.

C938 (R922) An *external-file-unit* shall be specified; if the optional characters UNIT= are omitted from the unit specifier, the unit specifier shall be the first item in the *wait-spec-list*.

C939 (R922) 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 WAIT statement.

The IOSTAT=, ERR=, EOR=, END=, and IOMSG= specifiers are described in 9.9.

The value of the variable specified in the ID= specifier shall be the identifier of a pending data transfer operation for the specified unit. If the ID= specifier is present, a wait operation for the specified data transfer operation is performed. If the ID= specifier is omitted, wait operations for all pending data transfers for the specified unit are performed.

Execution of a WAIT statement specifying a unit that does not exist, has no file connected to it, or was not opened for asynchronous input/output is permitted, provided that the WAIT statement has no ID= specifier; such a WAIT statement does not cause an error or end-of-file condition to occur.

9.6.2 Wait operation

A wait operation terminates a pending data transfer operation. Each wait operation terminates only a single data transfer operation, although a single statement may perform multiple wait operations.

If the actual data transfer is not yet complete, the wait operation first waits for its completion. If the data transfer operation is an input operation that completed without error, the storage units of the input/output storage sequence then become defined with the values as described in 9.5.1.14 and 9.5.4.4.

If any error, end-of-file, or end-of-record conditions occur, the applicable actions specified by the IOSTAT=, IOMSG=, ERR=, END=, and EOR= specifiers of the statement that performs the wait operation are taken.

If an error or end-of-file condition occurs during a wait operation for a unit, the processor performs a wait operation for all pending data transfer operations for that unit.

NOTE 9.48

Error, end-of-file, and end-of-record conditions may be raised either during the data transfer statement that initiates asynchronous input/output, a subsequent asynchronous data transfer statement for the same unit, or during the wait operation. If such conditions are raised during a data transfer statement, they trigger actions according to the IOSTAT=, ERR=, END=, and EOR= specifiers of that statement; if they are raised during the wait operation, the actions are in accordance with the specifiers of the statement that performs the wait operation.

After completion of the wait operation, the data transfer operation and its input/output storage sequence are no longer considered to be pending.

9.7 File positioning statements

R923 *backspace-stmt* **is** BACKSPACE *external-file-unit*
or BACKSPACE (*position-spec-list*)

R924 *endfile-stmt* **is** ENDFILE *external-file-unit*
or ENDFILE (*position-spec-list*)

R925 *rewind-stmt* **is** REWIND *external-file-unit*
or REWIND (*position-spec-list*)

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

R926 *position-spec* **is** [UNIT =] *external-file-unit*
or IOMSG = *iomsg-variable*
or IOSTAT = *scalar-int-variable*
or ERR = *label*

C940 (R926) No specifier shall appear more than once in a given *position-spec-list*.

C941 (R926) An *external-file-unit* shall be specified; if the optional characters UNIT= are omitted from the unit specifier, the unit specifier shall be the first item in the *position-spec-list*.

C942 (R926) 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.

The IOSTAT=, ERR=, and IOMSG= specifiers are described in 9.9.

Execution of a file positioning statement performs a wait operation for all pending asynchronous data transfer operations for the specified unit.

9.7.1 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 9.49

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 9.50

An example of a BACKSPACE statement is:

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

9.7.2 ENDFILE statement

Execution of an ENDFILE statement for a file connected for sequential access 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 for a file connected for sequential access, 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 connected for stream access causes the terminal point of the file to become equal to the curent file position. Only file storage units before the current position are considered to have been written; thus only those file storage units may be subsequently read. Subsequent stream output statements may be used to write further data to the file.

Execution of an ENDFILE statement for a file that is connected but does not exist creates the file; if the file is connected for sequential access it is created prior to writing the endfile record.

NOTE 9.51

An example of an ENDFILE statement is:

```
ENDFILE K
```

9.7.3 REWIND statement

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

NOTE 9.52

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 9.53

An example of a REWIND statement is:

```
REWIND 10
```

9.8 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.

R927 *inquire-stmt* **is** INQUIRE (*inquire-spec-list*)
or INQUIRE (IOLENGTH = *scalar-int-variable*) *output-item-list*

NOTE 9.54

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)
```

9.8.1 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:

R928 *inquire-spec* **is** [UNIT =] *external-file-unit*
or FILE = *file-name-expr*
or ACCESS = *scalar-default-char-variable*
or ACTION = *scalar-default-char-variable*
or ASYNCHRONOUS = *scalar-default-char-variable*
or BLANK = *scalar-default-char-variable*
or DECIMAL = *scalar-default-char-variable*
or DELIM = *scalar-default-char-variable*
or DIRECT = *scalar-default-char-variable*
or ERR = *label*
or EXIST = *scalar-default-logical-variable*
or FORM = *scalar-default-char-variable*
or FORMATTED = *scalar-default-char-variable*
or ID = *scalar-int-variable*
or IOMSG = *iomsg-variable*
or IOSTAT = *scalar-int-variable*
or NAME = *scalar-default-char-variable*
or NAMED = *scalar-default-logical-variable*
or NEXTREC = *scalar-int-variable*
or NUMBER = *scalar-int-variable*
or OPENED = *scalar-default-logical-variable*
or PAD = *scalar-default-char-variable*
or PENDING = *scalar-default-logical-variable*
or POS = *scalar-int-variable*

or POSITION = *scalar-default-char-variable*
or READ = *scalar-default-char-variable*
or READWRITE = *scalar-default-char-variable*
or RECL = *scalar-int-variable*
or ROUND = *scalar-default-char-variable*
or SEQUENTIAL = *scalar-default-char-variable*
or SIGN = *scalar-default-char-variable*
or SIZE = *scalar-int-variable*
or STREAM = *scalar-default-char-variable*
or UNFORMATTED = *scalar-default-char-variable*
or WRITE = *scalar-default-char-variable*

C943 (R928) No specifier shall appear more than once in a given *inquire-spec-list*.

C944 (R928) An *inquire-spec-list* shall contain one FILE= specifier or one UNIT= specifier, but not both.

C945 (R928) 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*.

C946 (R928) If an ID= specifier is present, a PENDING= specifier shall also be present.

The value of *external-file-unit* shall be nonnegative or equal to one of the named constants INPUT_UNIT, OUTPUT_UNIT, or ERROR_UNIT of the ISO_FORTRAN_ENV intrinsic module (13.12.1).

When a returned value of a specifier other than the NAME= specifier is of type character, 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=, ERR=, and IOMSG= specifiers are described in 9.9.

9.8.1.1 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. The interpretation of case is processor dependent.

9.8.1.2 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, DIRECT if the file is connected for direct access, or STREAM if the file is connected for stream access. If there is no connection, it is assigned the value UNDEFINED.

9.8.1.3 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.

9.8.1.4 ASYNCHRONOUS= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the ASYNCHRONOUS= specifier is assigned the value YES if the file is connected and asynchronous input/output on the unit is allowed; it is assigned the value

NO if the file is connected and asynchronous input/output on the unit is not allowed. If there is no connection, the *scalar-default-char-variable* is assigned the value UNDEFINED.

9.8.1.5 BLANK= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the BLANK= specifier is assigned the value ZERO or NULL, corresponding to the blank interpretation mode in effect for a connection 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.

9.8.1.6 DECIMAL= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the DECIMAL= specifier is assigned the value COMMA or POINT, corresponding to the decimal edit mode in effect for a connection 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.

9.8.1.7 DELIM= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the DELIM= specifier is assigned the value APOSTROPHE, QUOTE, or NONE, corresponding to the delimiter mode in effect for a connection 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.

9.8.1.8 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.

9.8.1.9 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.

9.8.1.10 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.

9.8.1.11 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.

9.8.1.12 ID= and PENDING= specifiers in the INQUIRE statement

The PENDING= specifier is used to determine whether or not previously pending asynchronous data transfers are complete. A data transfer operation is previously pending if it is pending at the beginning of execution of the INQUIRE statement.

The value of the variable specified in the ID= specifier shall be the identifier of a pending data transfer operation for the specified unit. If an ID= specifier is present and the specified data transfer operation is complete, then the variable specified in the PENDING= specifier is assigned the value false and the INQUIRE statement performs the wait operation for the specified data transfer.

If the ID= specifier is omitted and all previously pending data transfer operations for the specified unit are complete, then the variable specified in the PENDING= specifier is assigned the value false and the INQUIRE statement performs wait operations for all previously pending data transfers for the specified unit.

In all other cases, the variable specified in the PENDING= specifier is assigned the value true and no wait operations are performed; in this case the previously pending data transfers remain pending after the execution of the INQUIRE statement.

NOTE 9.55

The processor has considerable flexibility in defining when it considers a transfer to be complete. Any of the following approaches could be used:

(1) The INQUIRE statement could consider an asynchronous data transfer to be incomplete until after the corresponding wait operation. In this case PENDING= would always return true unless there were no previously pending data transfers for the unit.

(2) The INQUIRE statement could wait for all specified data transfers to complete and then always return false for PENDING=.

(3) The INQURE statement could actually test the state of the specified data transfer operations.

9.8.1.13 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 9.56

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.

The case of the characters assigned to *scalar-default-char-variable* is processor dependent.

9.8.1.14 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.

9.8.1.15 NEXTREC= specifier in the INQUIRE statement

The *scalar-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-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-int-variable* becomes undefined. If there are pending data transfer operations for the specified unit, the value assigned is computed as if all the pending data transfers had already completed.

9.8.1.16 NUMBER= specifier in the INQUIRE statement

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

9.8.1.17 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.

9.8.1.18 PAD= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the PAD= specifier is assigned the value YES or NO, corresponding to the pad mode in effect for a connection 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.

9.8.1.19 POS= specifier in the INQUIRE statement

The *scalar-int-variable* in the POS= specifier is assigned the number of the file storage unit immediately following the current position of a file connected for stream access. If the file is positioned at its terminal position, the variable is assigned a value one greater than the number of the highest-numbered file storage unit in the file. If the file is not connected for stream access or if the position of the file is indeterminate because of previous error conditions, the variable becomes undefined.

9.8.1.20 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.

9.8.1.21 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.

9.8.1.22 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.

9.8.1.23 RECL= specifier in the INQUIRE statement

The *scalar-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 file storage units. If there is no connection, or if the connection is for stream access, the *scalar-int-variable* becomes undefined.

9.8.1.24 ROUND= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the ROUND= specifier is assigned the value UP, DOWN, ZERO, NEAREST, COMPATIBLE, or PROCESSOR_DEFINED, corresponding to the I/O rounding mode in effect for a connection 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. The processor shall return the value PROCESSOR_DEFINED only if the I/O rounding mode currently in effect behaves differently than the UP, DOWN, ZERO, NEAREST, and COMPATIBLE modes.

9.8.1.25 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.

9.8.1.26 SIGN= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the SIGN= specifier is assigned the value PLUS, SUPPRSS, or PROCESSOR_DEFINED, corresponding to the sign mode in effect for a connection 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.

9.8.1.27 SIZE= specifier in the INQUIRE statement

The *scalar-int-variable* in the SIZE= specifier is assigned the size of the file in file storage units. If the file size cannot be determined, the variable is assigned the value -1.

For a file that may be connected for stream access, the file size is the number of the highest-numbered file storage unit in the file.

For a file that may be connected for sequential or direct access, the file size may be different from the number of storage units implied by the data in the records; the exact relationship is processor-dependent.

9.8.1.28 STREAM= specifier in the INQUIRE statement

The *scalar-default-char-variable* in the STREAM= specifier is assigned the value YES if STREAM is included in the set of allowed access methods for the file, NO if STREAM 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 STREAM is included in the set of allowed access methods for the file.

9.8.1.29 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.

9.8.1.30 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.

9.8.2 Restrictions on inquiry specifiers

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.

9.8.3 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-int-variable* in the IOLENGTH= specifier is assigned the processor-dependent number of file storage units that would be required to store the data of the output list in an unformatted file. 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.

The output list in an INQUIRE statement shall not contain any derived type list items that require a user-defined derived type input/output procedure as described in section 9.5.2. If a derived type list item appears in the output list, the value returned for the IOLENGTH specifier assumes that no user-defined derived type input/output procedure will be invoked.

9.9 Error processing

Error processing takes place if an error occurs during execution of an input/output statement that has an IOSTAT= or ERR= specifier, an end-of-file condition occurs during execution of an input/output statement that has an IOSTAT= or END= specifier, or an end-of-record condition occurs during execution of an input/output statement that has an IOSTAT= or EOR= specifier.

9.9.1 IOSTAT= specifier

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 the processor-dependent negative integer value of the constant IOSTAT_END of the intrinsic module ISO_FORTRAN_ENV (13.12.2.1) if an end-of-file condition occurs and no error condition occurs, or

(4) With a processor-dependent negative integer value of the constant IOSTAT_EOR of the intrinsic module ISO_FORTRAN_ENV (13.12.2.2) if an end-of-record condition occurs and no error condition or end-of-file condition occurs.

NOTE 9.57

An end-of-file condition may occur only for sequential or stream input and an end-of-record condition may occur only for nonadvancing input (9.5.3).

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
```

9.9.2 IOMSG= specifier

If an error, end-of-file, or end-of-record condition occurs during execution of an input/output statement, the processor shall assign an explanatory message to *iomsg-variable*. If no such condition occurs, the processor shall not change the value of *iomsg-variable*.

9.9.3 ERR= specifier

If an error condition (9.5.3) occurs during execution of an input/output statement that contains an ERR= specifier

(1) Processing of the input/output list 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 as specified in 9.9.1,

(4) If the input/output statement also contains an IOMSG= specifier, *iomsg-variable* becomes defined as specified in 9.9.2,

(5) If the statement is a READ statement and it contains a SIZE= specifier, the variable becomes defined with an integer value (9.5.1.14), and

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

9.9.4 END= specifier

If an end-of-file condition (9.5.3) occurs and no error condition (9.5.3) occurs during execution of an input statement that contains an END= specifier

(1) Processing of the input list terminates,

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

(3) If the input statement also contains an IOSTAT= specifier, the variable specified becomes defined as specified in 9.9.1,

(4) If the input statement also contains an IOMSG= specifier, *iomsg-variable* becomes defined as specified in 9.9.2, and

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

9.9.5 EOR= specifier

If an end-of-record condition (9.5.3) occurs and no error condition (9.5.3) or end-of-file condition (9.5.3) occurs during execution of an input statement that contains an EOR= specifier

(1) If the pad mode has the value YES, the record is padded with blanks to satisfy the input list item (9.5.4.4.2) and corresponding data edit descriptor that requires more characters than the record contains,

(2) Processing of the input list 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 as specified in 9.9.1,

(5) If the input statement also contains an IOMSG= specifier, *iomsg-variable* becomes defined as specified in 9.9.2,

(6) If the input statement contains a SIZE= specifier, the variable becomes defined with an integer value (9.5.1.14), and

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

9.10 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.

An input/output statement that is executed while another input/output statement is being executed is called a recursive input/output statement.

A recursive input/output statement shall not identify an external unit except that a child data transfer statement may identify its parent data transfer statement external unit.

An input/output statement shall not cause the value of any established format specification to be modified.

A recursive input/output statement shall nor modify the value of any internal unit except that a recursive WRITE statement may modify the internal unit identified by that recursive WRITE statement.

The value of a specifier in an input/output statement shall not depend on any *input-item*, *io-implied-do variable*, or on the definition or evaluation of any other specifier in the *io-control-spec-list* or *inquire-spec-list* in that statement.

The value of any subscript or substring bound of a variable that appears in a specifier in an input/output statement shall not depend on any *input-item*, *io-implied-do variable*, or on the definition or evaluation of any other specifier in the *io-control-spec-list* or *inquire-spec-list* in that statement.

In a data transfer statement, the variable specified in an IOSTAT=, IOMSG=, or SIZE= specifier, if any, shall not be associated with any entity in the data transfer input/output list (9.5.2) 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=, IOMSG=, or 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*.

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.

NOTE 9.58

Restrictions in the evaluation of expressions (7.1.8) prohibit certain side effects.