To: /B subgroup
Subject: Comments on section 9
From: Van Snyder

1 Edits

Edits refer to 01-007r2. Page and line numbers are displayed in the margin. Absent other instructions, a page and line number or line number range implies all of the indicated text is to be replaced by immediately following text, while a page and line number followed by + (-) indicates that immediately following text is to be inserted after (before) the indicated line. Remarks are noted in the margin, or appear between [and] in the text.

[There's no verb after the semicolon. Editor: Either replace the semicolon by a comma, or insert "they may be prepared" before "by some".] 161:39

[The note, as written, seems not to be germane at this point. Editor: remove the reference to (9.10) and move the note to [204:15+].] 162:30-32

[Is it really true that the name of a named file *is* a character string? Editor: Insert "the value of" after the last "is" on this line.] 162:30-33

[Editor: "the file" ⇒ "a file".] 163:14

[Record numbers are defined only for direct access files; they are not defined for sequential files. Editor: "record numbers" ⇒ "positions in the file".] 163:18

[The sentence "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" has unnecessarily convoluted structure. It's also not clear to what "this" refers. Replace the sentence by "If a READ statement is permitted for a connection, and a record has been written since the file was created, the record may be read from the file while it is connected to a unit."] 164:8-11

[Editor: To avoid confusion about the meaning of "unit" insert "file storage" after "preceeding".] 164:23

[There may be files for which it is *not* possible to identify the positions of file storage units – for example, a device such as an analog-to-digital converter. Editor: "Each" ⇒ "For a processor-dependent set of files, each".] 164:21

[There may be connections for which it is *not* permitted to specify the position of storage units to be read or written – for example, a connection to a device such as an analog-to-digital converter. Editor: "File" ⇒ "For a processor-dependent set of files, a connection may allow that file".] 164:24

[The sentence "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" has unnecessarily convoluted structure. It's also not clear to what "this" refers. Replace the sentence by "If a READ statement is permitted for a connection, and a file storage unit has been written since the file was created, the file storage unit may be read from the file while it is connected to a unit."] 164:26-30

[There may be files for which it is *not* possible to identify the positions of file storage units – for example, a device such as a serial port. Editor: "Each" ⇒ "For a processor-dependent set 164:21

of files, each".]

[There may be files for which it is *not* possible to identify the positions of file storage units – for example, a device such as a serial port (see 9.5.1.10). Editor: "Each" ⇒ "For a processor-dependent set of files, each".] 164:42

[Could be construed to specify that POS= specifiers in INQUIRE statements change file position.] 164:46-47

(6) For a processor-dependent set of files, a position that has been identified by a POS= specifier in an INQUIRE statement may be used to set the file position.

165:5+

NOTE 9.9$\frac{1}{2}$

An example of a file for which it is not possible to identify the file position is a device such as a serial port.

[The subclause applies only to sequential access connections. Editor: insert "connected for sequential access" after "file".] 165:7

[If the subclause somehow has application to direct access connections, the phrase "number of records" is incorrect. Editor: if the subclause applies for direct access connections, "in the file" ⇒ "in a file connected for sequential access, or the largest record number in a file connected for direct access".] 165:14

[Putting "always" and "unless" in the same sentence is inconsistent. Editor: Delete "always".] 165:26-27

[Putting "always" and "unless" in the same sentence is inconsistent. Editor: Delete "always".] 167:34-35

[The terms used here are not the names of the specifiers for the changeable modes, so it takes some investigation to discover that they're all covered. The initial values of changeable modes are the same that would result by not specifying the mode in an OPEN statement. It's simpler to say that, and it would cover all changeable modes, even if we add a few later. By specifying how padding works, the present wording appears to preclude the possibility that a changeable mode might be changed during a READ or WRITE statement.] 167:37-42

(8) The initial value of a changeable mode (9.4.1) is the value that would result from executing an OPEN statement that does not have a specification for the mode.

[Editor: Re-number subsequent items (I presume Frame does this automatically).]

[This restriction isn't quite right. One can imagine, for example, creating a file name by writing to an internal file, and then using the variable that is the internal file in a FILE= specifier in an INQUIRE or OPEN statement. Editor: After "specified" insert "as the unit". It may seem bizarre to do this, because the syntax doesn't allow an *internal-file-unit*, but an *external-file-unit* in an input/output statement in a user-defined derived-type input/output procedure might denote an internal file (that's the topic of issue 339).] 167:45-46

[The term "unit specifier" is used in the discussion of each input/output statement, but is not defined. Editor: Add a new sentence in the same paragraph:] 168:2

A **unit specifier** specifies the unit upon which an input/output statement ac. It itss of the form [UNIT=] *io-unit*, or more frequently [UNIT=] *external-file-unit* [*int-file-unit* if 01-270 passes].

A value of the *scalar-int-expr* that is nonnegative or the same as one of the named constants INPUT_UNIT, OUTPUT_UNIT or ERROR_UNIT from the ISO_FORTRAN_ENV intrinsic module (13.12.1) identifies the same external unit in all program units of the program. 168:32-33

NOTE 9.12$\frac{1}{2}$

> A unit argument of a user-defined derived-type input/output procedure that does not meet the requirements of the above paragraph may be used to refer to an internal file in a parent data transfer statement.

[The term "permanently" is a bit extreme.] 169:17-18
The modes of a connection to an external file remain in effect for the duration of that connection, or until they are changed by a subsequent OPEN statement that modifies the connection.

[Editor: "is present" ⇒ "appears". The term "present" applies to optional arguments.] 170:37

[Duplicates part of [169:39-40]. There's no point to duplication, and a half-truth is misleading. Editor: Delete.] 171:5-6

[Editor: "be present" ⇒ "appear" twice.] 171:35-36

[The phrases "for an existing file" and "for a new file" are used in the discussion of several specifiers. It's not clear how this interacts with a STATUS= specifier with the value REPLACE.] 172:1–

In subsequent discussions of specifiers, the term "existing file" refers to a file that exists at the moment the connection is established, and the term "new file" refers to a file that does not exist at the moment the connection is established.

NOTE 9.18$\frac{1}{3}$

> A file may exist at the moment the OPEN statement begins execution, but cease to exist before the connection is established. This may occur, for example, if a STATUS= specifier appears and has the value REPLACE.

[**PROPOSED SPEC CHANGE** to help some of our European colleagues: Editor: "POINT" ⇒ "processor dependent", and then add a note:] 172:31

NOTE 9.18$\frac{2}{3}$

> The processor could provide the initial value in any of several ways. For example, by reference to an environment variable, an operating system's localization facilities, a configuration file, a command-line option specified when the program is executed, or a settings notebook.

[Why does this say "A file that did not exist previously (a new file, either specified explicitly or by default)" while elsewhere it just says "A new file"? Editor: "A file ... default)" ⇒ "A new file".] 173:19-20

[Editor: "be present" ⇒ "appear" twice.] 173:29-30

[Editor: "be present" ⇒ "appear"; "is present" ⇒ "appears".] 176:30-31

[Editor: "is present" ⇒ "appears".] 176:41

[Editor: Delete. (What else might an asterisk signify here?)] 176:43

[Editor: "be present" ⇒ "appear".] 176:44

[Editor: "is present" ⇒ "appears" twice.] 176:47-48

[Editor: "is present" ⇒ "appears" twice.] 177:3,5

[Editor: "is present" ⇒ "appears" twice.] 177:7,9

[Editor: "be present" ⇒ "appear" twice.] 177:11,13

[Editor: Alphabetize the paragraphs that describe the specifiers in input/output statements.]	9.5.1
[Editor: "is present" ⇒ "appears".]	178:4
[There is a definition of the meaning of "identifier" here. Editor: Embolden "identifier" and add it to the index.]	179:19
[Editor: "be present" ⇒ "appear".]	179:29
[Editor: "be present" ⇒ "appear".]	180:3
[Editor: insert "only" after "list".]	181:11
[Editor: "allocatables" ⇒ "allocatable" twice.]	181:37,43
[Editor: "An" ⇒ "Otherwise, an". Then move to [181:37+]. Otherwise it conflicts with [181:32-37].]	182:8-20
[Editor: Insert "is" after "record".]	182:35
[Editor: "item" ⇒ "items"; "descriptor" ⇒ "descriptors", but don't bother to do it if subsection 1.1 passes, because it deletes the text to which this applies.]	183:10
[Editor: insert "9.5.3 and" before "9.9" but don't do this if subsection 1.1 passes, because those edits incorporate 9.5.3 into 9.9.]	183:44
[Editor: insert "9.5.3 and" before "9.9" but don't do this if subsection 1.1 passes, because those edits incorporate 9.5.3 into 9.9.]	184:22
[Editor: It appears that there's an extra blank after "either".]	184:33
[Editor: "is not present" ⇒ "does not appear".]	184:34
[Editor: Insert "It need not exist." as a new sentence.]	185:17
[Editor: "is present" ⇒ "appears".]	185:20
[Editor: Delete "The next effective item to be processed is called the next effective item" because it doesn't say anything.]	185:31-32
[Editor: "theh" ⇒ "the".]	187:23
[User-defined derived-type input/output does not alter the processing of derived-type effective items – it completely replaces it. Editor: "alter" ⇒ "override".]	187:33
[Editor: "as" ⇒ "that is".]	187:34
[Editor: "list" ⇒ "effective".]	188:34
[Editor: "is present" ⇒ "appears".]	191:19
[Editor: Move [191:39-42] to here.	191:23+
[Editor: "values" ⇒ "a value".]	191:27
[Editor: ".TRUE." ⇒ "true".]	191:32
[Editor: "is present" ⇒ "appears".]	193:35
There is a processor dependent set of files for which file positioning statements are permitted.	194:23+

NOTE 9.48$\frac{1}{3}$

An example of a file for which file positioning is not permitted is a device such as a serial port.

or END= *label* 194:34+
or EOR= *label*

[to cover the case that the end-of-file or end-of-record is detected by a wait operation performed by the file positioning statement. Editor: Move [194:37] to here, to alphabetize the list.]

[Editor: If "there is no current record and no preceding record" is just a fancy of saying "the file is at its initial point" replace the former by the latter.] 195:4

195:5+

NOTE 9.48$\frac{2}{3}$

> If the last data transfer statement executed for the file was a formatted data transfer statement that transferred several records, a BACKSPACE statement positions the file before the last of them, not before all of them.

[There is no good reason why backspacing over records written using list-directed or namelist formatting is prohibited. Records are perfectly well defined in these cases. The problem is that one might not know where a record boundary is. But this also applies in the case of "ordinary" formatted output that has / edit descriptors or in which format reversion takes place.] 195:11

NOTE 9.49$\frac{1}{2}$

> Backspacing over records written by list-directed or namelist formatting is probably not a worthwhile thing to try to do. List-directed or namelist formatting can put record boundaries in unpredictable places.

---

or END= *label* 196:31+
or EOR= *label*

[to cover the case that the end-of-file or end-of-record is detected by a wait operation performed by the INQUIRE statement. Editor: Move [196:24] to here, to alphabetize the list (like it was done for OPEN).]

[UNIT= is optional. Editor: "UNIT=" ⇒ "unit".] 197:13

[Editor: "is present" ⇒ "appears"; "be present" ⇒ "appear".] 197:18

[Editor: "variable" ⇒ "variables"; "specifier" ⇒ "or IOMSG= specifiers".] 197:25

[Editor: insert "(9.5.1.5, 10.7.6)" after "mode".] 198:5

[Editor: insert "(9.5.1.6, 10.7.8)" after "mode".] 198:10

[Editor: insert "(9.5.1.7, 10.9.2, 10.10.2.1)" after "mode".] 198:15

[Editor: "is present" ⇒ "appears".] 199:2

[How do you read "on" a file? Editor: "or written on" ⇒ "from or written to".] 199:39

[Editor: insert "(9.5.1.9, 9.5.4.4.2)" after "mode".] 200:12

[There may be files for which it is *not* possible to identify the positions of file storage units – for example, a device such as a serial port (see 9.5.1.10). Editor: "The" ⇒ "For a processor-dependent set of files, the".] 200:16

[Editor: insert "(9.5.1.12, 10.7.7)" after "mode".] 201:10

[Editor: "than" ⇒ "from".] 201:14

[Editor: "SUPPRSS" ⇒ "SUPPRESS".] 201:23

[Editor: insert "(9.5.1.13, 10.7.4)" after "mode".]	201:24
[UNIT= is optional. Editor: "UNIT=" ⇒ "unit" twice.]	202:10-11
[Since an inquire by output list form "includes only an IOLENGTH= specifier" there's no point in saying that it "does not include a FILE= or UNIT= specifier." Editor: Delete "does not include a FILE= or UNIT= specifier and".]	202:15-16
[Editor: In most references, "section" is capitalized. Capitalize it here.]	202:23
[Editor: "in" ⇒ "on".]	204:44

1.1 Subclauses 9.5.3 and 9.9 ought to be combined

Subclauses 9.5.3 and 9.9 contain some duplicative material, some contradictory material and some incomplete material (mostly incomplete with respect to wait operations). There is no subclause not in 9.5 that addresses error conditions, but error conditions can occur in other than data transfer statements. To address these problems, subclauses 9.5.3 and 9.9 ought to be combined.

This subsection moots some edits in the main part of section 1.

[Editor: Delete.]	183:1-23
[Editor: "9.5.3" ⇒ "9.9" twice.]	166:11,12
[Editor: "9.5.3" ⇒ "9.9".]	166:27
[Editor: "9.5.3" ⇒ "9.9".]	183:43
[Editor: "9.5.3" ⇒ "9.9".]	184:21
[Editor: "9.5.3" ⇒ "9.9".]	187:18
[Editor: "9.5.3" ⇒ "9.9" twice.]	191:25,37
[Editor: "9.5.3" ⇒ "9.9".]	193:8
[Editor: Move [182:31-183:7] to replace the subclause heading and this paragraph. Make sure to do the edit specified above for [182:35] first. This paragraph is useless and repetitive introductory waffle, so it's not a loss to just replace it.]	202:26-30
[Editor: Delete "(9.5.3)" twice.]	203:3,18
[Editor: Delete "(9.5.3)" twice.]	203:30
[Editor: Delete "(9.5.3)" thrice.]	203:41-42

9.9.3 Error conditions and the ERR= specifier

203:17-19

If an error condition occurs during execution of a data transfer the position of the file becomes indeterminate. [This is here because it occurs independently of whether the statement contains an ERR= or an IOSTAT= specifier. The lack of "statement" after "transfer" is intentional, to cover the case of an error condition occurring during a wait operation.]

If an error condition occurs during execution of an input/output statement that contains neither an ERR= specifier nor an IOSTAT= specifier, execution of the program is terminated. If an error condition occurs during execution of an input/output statement that contains either an ERR= specifier or an IOSTAT= specifier

[Editor: Insert ", if any," after "list".]	203:20

(2) If the statement is a data transfer statement or the error condition occurs during a wait operation, all implied DO variables in the statement that initiated the transfer become undefined. 203:21

($5\frac{1}{2}$) If the statement is a READ statement or the error condition occurs in a wait operation for a transfer initiated by a READ statement, all input list items or namelist group objects in the statement that initiated the transfer become undefined. [Is the "in" part good enough to cover namelist group objects?] 203:27+

[Editor: "Execution" ⇒ "If an ERR= specifier appears, execution".] 203:28

9.9.4 End-of-file conditions and the END= specifier 203:29-31

If an end-of-file condition occurs during execution of an input/output statement that contains neither an END= specifier nor an IOSTAT= specifier, execution of the program is terminated. Otherwise if an error condition does not occur [we don't need the "If an end-of-file condition occurs during execution of an input/output statement..." part that we needed for error conditions, because end-of-file conditions can only occur during execution of input/output statements]

($1\frac{1}{3}$) All implied DO variables in the statement that initiated the transfer become undefined. 203:32+

($1\frac{2}{3}$) If the statement is a READ statement or the end-of-file condition occurs in a wait operation for a transfer initiated by a READ statement, all input list items or namelist group objects in the statement that initiated the transfer become undefined.

[Editor: "input" ⇒ "input/output" twice to cover the case of an end-of-file condition occurring during a wait operation.] 203:35,37

[Editor: "Execution" ⇒ "If an END= specifier appears, execution".] 203:39

9.9.5 End-of-record conditions and the EOR= specifier 203:40-42

[This needs to be checked carefully because the material from which it is taken at [183:8-13] and [183:22-23] is difficult to follow, and may be contradictory.]

If an end-of-record condition occurs during a data transfer initiated by a nonadvancing input statement, the statement in which the condition is detected has neither an EOR= nor an IOSTAT= specifier, and the pad mode has the value NO, execution of the program is terminated. Otherwise if neither an error nor an end-of-file condition occurs

($2\frac{1}{2}$) If the statement that initiated the transfer is a nonadvancing statement, all implied DO variables in that statement become undefined. 204:4+

[Editor: "input" ⇒ "input/output" twice to cover the case of an end-of-record condition occurring during a wait operation.] 204:6,8

[Editor: "Execution" ⇒ "If an EOR= specifier appears, execution".] 203:39

[Editor: Move [202:31-203:16] to here.] 204:12+

[Editor: "9.5.3" ⇒ "9.9".] 385:24

2 Don't know what to do

This doesn't specify what happens if the value of the STATUS= specifier is SCRATCH and 174:18

the file exists. What do want to happen?	
This constraint is rather fatuous, given that one can do a nonadvancing asynchronous read, and then put an EOR= specifier in a WAIT statement.	176:47
The term “base object” is defined in 6.1.2. It’s not obvious that it’s defined for anything other than structures.	178:35
The explanation that a variable may be a pending input/output storage sequence affector in some situations and not others is schizophrenic. First it discusses different scoping units, and then different instants of time. Which is it?	179:1-3
Appears to contradict [183:8-13]. This paragraph is deleted by edits in subsection 1.1.	183:22-23
Because “entity” isn’t defined, it’s not clear here what happens if we have, e.g. `X` = 10,20,30 for a 3-element array, followed by `X`(2) = 100. Does all of `X` get its value from the last occurrence of `X`? Perhaps “entity” ought to be “variable”.	186:2
What does this mean? At [162:17] it says that an endfile record is the last record of a file. How can there be records other than those “written before the endfile record?” What if one writes an endfile record, backspaces over it, writes some more records, and writes a new endfile record? Are the intervening records accessible during a subsequent connection for direct access? This was in Fortran 95. Do we need an interp for it?	195:18-20
Does writing additional data after writing an endfile “record” to a file connected for stream access change the terminal point?	195:27-28
Is there a set of specifiers that is allowed for inquire by name but not inquire by unit, or vice-versa? If so, a summary here would be useful.	197:26+
Is this “when” correct?	202:20

3 Problem with file storage units

The specifications concerning file storage units may cause problems for some architectures. At [166:37-42] we see

> 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.

On some systems, values of some types are not commensurate with 8-bit octets. For example there have been and may again be systems that have 36- or 60-bit words. On these systems, on some devices, it is impossible (or at least extremely difficult) to put words at any position other than a word boundary. On other devices, it is difficult to put characters at other than character boundaries. Although the size of characters and words are both commensurate with 4-bit file storage units, it is a burden to allow either words or characters (depending on the device) to be put at arbitrary multiples of 4-bit positions.

The second paragraph above may provide so much latitude as to make for very inefficient data transfer on such systems. Rather than saying that it shall be possible to store a value at any integer multiple of a file storage unit, specify instead that no padding shall be required, but this does not require that it shall be possible to read or write a value from any position that is an integer multiple of a file storage unit.