# Fortran preprocessor requirements

INCITS/Fortran JoR

*[2023-10-24 Tue 11:27]*

## 1 Summary

We summarize the potential requirements for the Fortran 202y preprocessor.

Each heading is of the form `[shorthand-name] One-line description`. There may be an additional tag (category) on the line (such as `:output:`). The section body contains information on

- Where the requirement came from in Fortran discussions and posts
- Current status (TBD, accepted by JoR, accepted by WG5, rejected by JoR, rejected by WG5, etc.)
- Where the reference appears in the C 2018 standard *ISO/IEC 9899:2018 Information technology – Programming languages – C*. (We will update for 2024 when ratified.) We write these references with
  - an opening square bracket `[` and the letter `C`, followed by
  - the section marker `§` and the section number, followed by
  - (optionally) a `¶` followed by the paragraph number, followed by
  - (optionally) a bullet symbol · and the bullet number or letter, followed by
  - (optionally) the section title for redundant context, followed by
  - a closing square bracket `]`.

  For example, `[C§5.1.1.2¶1·2 Translation phases]` references section 5.1.1.2 of the C 2018 standard, paragraph 1, second bullet point. This is in the "Translation phases" section.
- Where the reference appears in the Fortran 2023 standard *ISO/IEC FDIS 1539-1 Information technology — Programming languages — Fortran*. We write these references with

- an opening square bracket `[` and the string `F2023`, followed by
- the section marker `§` and the section number, followed by one of
  - an `R` followed by a rule number, or
  - a `C` followed by a constraint number, or
  - a ¶ followed by the paragraph number, followed optionally by a bullet symbol · and the bullet number or letter
- (optionally) the section title for redundant context, followed by
- a closing square bracket `]`.

For example, `[F2019§13.2.1R1301 FORMAT statement]` references section 13.2.1 of the Fortran 2023 standard, rule 1301. This is in the "FORMAT statement" section.

## 1.1 Translation phases

Similar to the C standard, we define phases of text processing related to preprocessing. The preprocessor performs the following phases on the input source program before the processor transforms the program for use.

This requires implementations to behave as if these separate preprocessing phases occur, even though they are typically folded together in practice. Source input need not necessarily be stored as files, nor need there be any one-to-one correspondence between these entities and any external representation. The description is conceptual only, and does not specify any particular implementation.

1. Continuation lines in the input source are interpreted, producing a sequence of logical lines (introducing new-line characters for end-of-line indicators). [C§5.1.1.2¶1·1-2]

2. The preprocessor decomposes the logical lines into preprocessing tokens and sequences of white-space characters and comments. This input shall not end in a partial preprocessing token. New-line characters are retained. For the proper handling of compiler directives, comments are retained. Whether each nonempty sequence of white-space characters other than new-line is retained or replaced by one space character is implementation-defined. [C§5.1.1.2¶1·3]

3. Preprocessing directives are executed and macro invocations are expanded. A #include preprocessing directive causes the named header or source file to be processed from phase 1 through phase 3, recursively. All preprocessing directives are then deleted. [C§5.1.1.2¶1·4]

4. Each preprocessing token is converted into a token for subsequent handling by the processor.

# 2 Phase 1 Continuation handling

## 2.1 [c-cont] C-style line continuations in directives

- **Status:** TBD
- **Source:** Fla1, [C§5.1.1.2¶1·2 Translation phases]

In fixed-form and free-from source code, delete a backslash \ immediately followed by a newline character.

From The C standard:

> Each instance of a backslash character (\) immediately followed by a new-line character is deleted, splicing physical source lines to form logical source lines. Only the last backslash on any physical source line shall be eligible for being part of such a splice. A source file that is not empty shall end in a new-line character, which shall not be immediately preceded by a backslash character before any such splicing takes place.

## 2.2 [fortran-cont-fixed] Process Fortran line continuation in directives

- **Status:** TBD
- **Source:** Che1

In fixed-form input, a character in column 6 that is not blank or the digit zero continues the line with the previous line, even if the previous line is a directive line, or the continuation of a directive line.

## 2.3 [fortran-cont-free] Fortran line continuation in directives in free form

- **Status:** TBD
- **Source:** Che1, [C§6.10 Preprocessing directives]

In free-form input, an `&` character as the last character on a directive line indicates the directive continues on the next line.

### 2.4 [fortran-cont-free-trim] Fortran line continuation in directives in free form

- **Status:** TBD
- **Source:** Che1, [C§6.10 Preprocessing directives]

In free-form input, an `&` character as the last character on a directive line indicates the directive continues on the next line. When the first non-blank character on the next line is also an `&`, the characters between the ampersands are deleted.

### 2.5 [c-comment-strip] Strip C-style `/* ... */` comments

- **Status:** TBD
- **Source:** Fla1, [C§6.10 Preprocessing directives]

### 2.6 [comment-definition-cont] Comment lines in definitions with continuation lines

- **Status:** TBD
- **Source:** Fla1, [C§6.10 Preprocessing directives]

### 2.7 [comment-bang] Recognize the comment character '!' in directives

- **Status:** TBD
- **Source:** Che1, [C§6.10 Preprocessing directives]

## 3 Phase 2 Tokenization

### 3.1 [tokens-case-insensitive] Case insensitive tokens

- **Status:** TBD
- **Source:** Fla1

Fortran is not case-sensitive. The preprocessor is not be case-sensitive when recognizing identifiers. The text fragment

```
#define abc XYZ
#define ABC foo
      subroutine abc
```

should expand to

```
      subroutine foo
```

If the preprocessor were case sensitive, we would have behavior like

```
#define ABC var_1
#define abc var_2
      abc = ABC + 1     ! Normally, Fortran treats these as the same identifier
```

expanding to

```
      var_2 = var_1 + 1    ! These identifiers are now different
```

We should expect it to expand to

```
      var_2 = var_2 + 1    ! Only the second definition matters
```

### 3.2 [spaces-end-token] Spaces significant in determining tokens

- **Status:** TBD
- **Source:** Fla1

In order to simplify the preprocessor tokenization, spaces are significant, even in fixed-form source.

## 4 Phase 3 Directive processing

### 4.1 [non-directive] # non-directive

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.2 [#if] # if

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.3 [#ifdef] # ifdef

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.4 [#ifndef] # ifndef

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.5 [#elif] # elif

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.6 [#else] # else

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.7 [#endif] # endif

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.8 [#include] # include

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.9 [#include-computed] # include (computed)

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.10 [#define-id] # define id replacement-list

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.11 [#define-id-function] # define id ( id-list ) replacement-list

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.12 [#define-id-0-varargs] # define id ( ... ) replacement-list

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.13 [#define-id-n-varargs] # define id ( id-list , ... ) replacement-list

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.14 [#undef] # undef

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.15 [#line] # line

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.16 [#error] # error

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.17 [#pragma] # pragma

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.18 [#newline] # new-line

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 4.19 [#show] # show

- **Status:** TBD
- **Source:** Lio3, [C§6.10 Preprocessing directives]

### 4.20 [#import] # import VARNAME

- **Status:** TBD
- **Source:** Lio3, [C§6.10 Preprocessing directives]

### 4.21 [#output] # output filename [–append]

- **Status:** TBD
- **Source:** Lio3, [C§6.10 Preprocessing directives]

## 5 Expressions

### 5.1 [#-operator] #

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 5.2 [##-operator] ##

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 5.3 [defined-operator] `defined`

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 5.4 [bang-operator] `!`

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 5.5 [c-expressions] C-style expressions

- Source :::
- **Status:** TBD
- **C reference:** [C§6.10 Preprocessing directives]

### 5.6 [fortran-expressions] Fortran-style expressions

- Source :::
- **Status:** TBD
- **C reference:** [C§6.10 Preprocessing directives]

## 6 Expansion

### 6.1 [fixed-no-expand-c-col-1] No expansion of `C` (or `D`) in column 1

- **Status:** TBD
- **Source:** Ble1, [C§6.10 Preprocessing directives]

In fixed-form, `C` in column 1 indicates a comment. No identifier that begins with a `C` in column is expanded.

### 6.2 [fixed-no-expand-d-col-1] No expansion of `D` in column 1

- **Status:** TBD
- **Source:** Ble1, Fla1, [C§6.10 Preprocessing directives]

In fixed-form, `D` in column 1 is a common extension to indicate a comment. No identifier that begins with a `D` in column is expanded.

### 6.3 [fixed-no-expand-col-6] No expansion of column 6

- **Status:** TBD
- **Source:** Kli1, [C§6.10 Preprocessing directives]

In fixed-form, a character in column 6 that is not blank or zero indicates a continuation line. No identifier that begins in column 6 is expanded.

[Note that since we define expansion to occur after continuation handling, this requirement is not necessary.]

### 6.4 [fixed-strip-col-1-comments] Strip column 1 `C` comments from expanded text

- **Status:** TBD
- **Source:** Fla1, [C§6.10 Preprocessing directives]

### 6.5 [no-expand-string] No expansion in strings

- **Status:** TBD
- **Source:** Ble1, Fla1, [C§6.10 Preprocessing directives]

String constants are output without being examined for macro expansion.

### 6.6 [no-expand-hollerith] No expansion in Hollerith

- **Status:** TBD
- **Source:** Ble1, [C§6.10 Preprocessing directives]

No expansion occurs in the string contained in a Hollerith constant.

### 6.7 [no-expand-implicit-char-list] No expansion in `IMPLICIT` single-character specifiers

- **Status:** TBD
- **Source:** Ble1, [C§6.10 Preprocessing directives]

The letters in an `IMPLICIT` statement are not considered for macro expansion.

[Note that this implies the preprocessor recognizes `IMPLICIT`.]

### 6.8 [no-expand-format] No expansion of `FORMAT` specifiers

- **Status:** TBD
- **Source:** Ble1, Fla1, [F20123§13.2.1R1301]

In `FORMAT` statements, there is no macro expansion in the *format-specification.*.

[Note that this implies the preprocessor recognizes `FORMAT` statements.]

### 6.9 [expand-comments] Expansion in comments

- **Status:** TBD
- **Source:** Ble1, [C§6.10 Preprocessing directives]

### 6.10 [expand-directives] Expansion in directives (e.g., OpenMP)

- **Status:** TBD
- **Source:** Ble1, [C§6.10 Preprocessing directives]

### 6.11 [preprocess-fortran-include] Expand `INCLUDE` lines as if `#include`

- **Status:** TBD
- **Source:** Fla1, Jor1, [C§6.10 Preprocessing directives]

## 7 Output form

### 7.1 [fixed-clip-input] Right margin clipping at column 72

- **Status:** TBD
- **Source:** Fla1, [C§6.10 Preprocessing directives]

### 7.2 [fixed-no-directive-clip] No right margin clipping on directive lines

- **Status:** TBD
- **Source:** Fla1, [C§6.10 Preprocessing directives]

### 7.3 [fixed-output-conform] Expanded text reflects fixed-format rules

- **Status:** TBD
- **Source:** Fla1

## 8 Predefined macros

### 8.1 [file-process-date] `__DATE__`

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 8.2 [file-name-context] `__FILE__`

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 8.3 [line-number-context] `__LINE__`

- **Status:** TBD
- **Source:** cpp, [C§6.10 Preprocessing directives]

### 8.4 [fortran-conform] `__STDFORTRAN__`

- **Status:** TBD
- **Source:** cpp-ish, [C§6.10 Preprocessing directives]

### 8.5 [hosted-implementation] `__STDFORTRAN_HOSTED__`

- **Status:** Not accepted
- **Source:** cpp-ish, [C§6.10 Preprocessing directives]

### 8.6 [fortran-version] `__STDFORTRAN_VERSION__`

- **Status:** TBD
- **Source:** cpp-ish, [C§6.10 Preprocessing directives]

### 8.7 [file-process-time] `__TIME__`

- Source :::
- **Status:** TBD
- **C reference:** [C§6.10 Preprocessing directives]

### 8.8 [stringify-macro] `STRINGIFY`

- **Status:** Not accepted
- **Source:** Clu1, [C§6.10 Preprocessing directives]

### 8.9 [scope-macro] `__SCOPE__`

- **Status:** Not accepted
- **Source:** Clu1, Lio1, [C§6.10 Preprocessing directives]

### 8.10 [vendor-macro] `__VENDOR__`

- **Status:** Not accepted
- **Source:** Clu1, [C§6.10 Preprocessing directives]

### 8.11 [no-undecorated-std-definitions] undecorated names (no _) defined by preprocessor

- **Status:** TBD
- **Source:** Lio2, [C§6.10 Preprocessing directives]

## 9 Sources

- cpp: *cpp* if in the C standard (2018), *cpp-ish* if in C standard, but "Fortranized".
- Ble1: JoR Email threads from Rich Bleikamp re: tutorial *[2022-08-08 Mon 21:34]*.
- Che1: Email from Daniel Chen to JoR *[2022-07-29 Fri 11:08]*.
- Clu1: Email from Tom Clune *[2022-08-01 Mon 10:48]*.
- Fla1: LLVM Flang Preprocessing.md [`https://github.com/llvm/llvm-project/blob/main/flang/docs/Preprocessing.md`]
- Jor1: JoR meeting on preprocessors *[2022-08-22 Mon 10:00]*.
- Jor2: JoR meeting on preprocessors *[2022-09-20 Tue 13:00]*.
- Kli1: Private communication in his head.
- Lio1: Email from Steve Lionel *[2022-08-01 Mon 13:52]*.
- Lio2: JoR discussion forum `https://j3-fortran.org/forum/viewtopic.php?p=561`
- Lio3: JoR discussion forum `https://j3-fortran.org/forum/viewtopic.php?p=562`

## 10 References


- Jor email re: cpp tutorial for October meeting?
- INCITS+ISO+IEC+9899+2018+(2019)
- LLVM Flang Preprocessing.md[ `https://github.com/llvm/llvm-project/blob/main/flang/docs/Preprocessing.md`