

# FFP Phase 4 – Preprocess Directives

INCITS/Fortran JoR

November 26, 2024

## Contents

<b>1</b>	<b>Introduction</b>	<b>2</b>
<b>2</b>	<b>Translation phases</b>	<b>2</b>
2.1	Phase 1: Remove continuations . . . . .	2
2.2	Phase 2: Process comments . . . . .	2
2.3	Phase 3: Tokenize the source into preprocessing tokens . . . . .	3
2.4	Phase 4: Execute preprocessor directives . . . . .	3
<b>3</b>	<b>High-level requirements</b>	<b>3</b>
<b>4</b>	<b>Detailed requirements</b>	<b>5</b>
4.1	Requirement sources . . . . .	5
4.2	Expansion . . . . .	6
4.2.1	No expansion of function macro names not followed by parenthesis . . . . .	6
4.2.2	Function macro invocation may cross logical line bound- aries . . . . .	6
4.2.3	No expansion of self-referential macro names . . . . .	6
4.2.4	No expansion in strings . . . . .	6
4.2.5	No expansion in Hollerith . . . . .	7
4.2.6	No expansion in <code>IMPLICIT</code> single-character specifiers . . . . .	7
4.2.7	No expansion in <code>FORMAT</code> specifiers . . . . .	7
4.2.8	Expand <code>INCLUDE</code> lines as if <code>#include</code> . . . . .	7
4.2.9	Expand macro names in <i>kind-param</i> in literal constants . . . . .	8
<b>5</b>	<b>Syntax</b>	<b>8</b>

## 1 Introduction

The INCITS/Fortran committee has discussed several times in the Plenary meetings, and the JoR subgroup has met several times, to lay down formal requirements for the proposed Fortran preprocessor (FPP) for Fortran 202y (presumably 2028).

This paper lists the requirements and syntax for Phase 4 of the preprocessor, which processes directives after Phase 3 tokenizes the input stream.

## 2 Translation phases

The C standard [ISO24] defines eight translation phases. These phases each perform a well-defined set of operations on the C source code and intermediate representations. They define a processing pipeline where one phase transforms its input in some way, and its output becomes the input to the next phase.

While these phase descriptions explain how C compilers *should behave*, they do not prescribe how C compilers *should be written*.

We do the same for Fortran. For FPP, though, we are only concerned with phases through interpreting preprocessor directives.

### 2.1 Phase 1: Remove continuations

For fixed-form Fortran source, follow the column-6 conventions to produce a sequence of logical lines.

For free-form Fortran source, follow the & conventions to produce a sequence of logical lines.

In either form, remove continuations from directive lines (those lines beginning with #).

### 2.2 Phase 2: Process comments

For fixed and free-form source, translate comment-based directives (such as `!dir$`, `!omp$`, `!acc$`, and `CDIR$`, `COMP$`, and `Cacc$`) into some kind of formal pragma (such as a `#pragma` directive). Replace other comments with spaces.

### 2.3 Phase 3: Tokenize the source into preprocessing tokens

The output from Phase 3 is converted to preprocessor tokens according to the rules defined in “On Fortran source form” above.

### 2.4 Phase 4: Execute preprocessor directives

Preprocessing directives in the output from Phase 4 are executed. As in C, the execution of preprocessor directives and interpretation of macro definition and expansion is a *token-replacement* process, not a *text replacement* process.

Macros are expanded in Fortran source.

Source code is included, excluded, or modified based on the directives.

## 3 High-level requirements

Steve Lionel, the ISO WG 5 (Fortran) Convenor, presented this list of high-level requirements at INCITS/Fortran meeting #232, February 2024 [Sub24]

Fortran programmers use preprocessing for several reasons:

- Adjusting external names for C interoperability (largely obviated by C interop features in the language)
- Platform/OS specific code
- Debug or other variants of code
- A crude way of implementing templates

Explicit goals for Fortran 202y were

- Define cpp-style preprocessor in the standard
- Ideally, most existing uses of preprocessing will “just work”, or need minimal changes
- Feature will not be optional, but implementations are encouraged to offer an option to “do it the old way”
- “Minimum Viable Product” – don’t try to do everything

Features should have the same semantics as the C preprocessor, except as noted.

- `__LINE__` and `__FILE__` defined macros
- `#line`
- `#ifdef`, `#ifndef`, `#endif`
- `#if`, `#elif`, `#else`, `#endif`
- `#define` and `#undef`
- `#include`
- `#error`
- `#` operator (character literal from token)
- `##` operator (token concatenation)
- `/* ... */` C-style comments are allowed in directives (including multi-line comments)
- `\` new-line continuations allowed in directives
- `#pragma` directive for implementation-specific directives
- Macro expansion
  - tokens are case-sensitive
  - tokens are not replaced in character literals
  - tokens are not replaced in Hollerith strings
  - tokens are not replaced in *letter-spec-list* of `IMPLICIT` statements
  - tokens are not replaced in column 6 in fixed-form source
  - tokens are replaced in comment-style directives (such as `!omp$`, `!dir$`)
- Expressions
  - `defined` operator
  - `//` is the Fortran concatenation operator; it does not introduce a C-style comment
  - `!` is the C “not” operator; it does not introduce a Fortran comment

## 4 Detailed requirements

We list the detailed requirements for handling preprocessor directives.

Each requirement is a heading with a “One-line description”, followed by a short table that contains a unique identifier, the requirement’s current status in the INCITS/Fortran approval process, and where the requirement came from (such as a C or Fortran standard, an INCITS meeting, or on email conversation).

[Note: Most of the detailed requirements here are still being worked...]

### 4.1 Requirement sources

These requirements came from the following sources.

**[C2018]** The C standard [\[ISO18\]](#).

**[C2024]** The C standard [\[ISO24\]](#).

**[F2023]** The Fortran standard [\[ISO23\]](#).

**[G14]** The C Preprocessor [\[SW24\]](#).

**facpp** The C standard, but made Fortran-aware.

**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]*.

**gak** Gary Klimowicz as he wrote these specifications.

**jor1** JoR meeting on preprocessors *[2022-08-22 Mon 10:00]*.

**jor2** JoR meeting on preprocessors *[2022-09-20 Tue 13:00]*.

**jor3** JoR meeting on preprocessors *[2023-11-07 Tue 12:00]*.

**jor4** JoR meeting on preprocessors *[2022-12-06 Tue 12:00]*.

**jor5** Preprocessor Take 2 presentation from Meeting #232 February 2024 [\[Sub24\]](#).

**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>.

## 4.2 Expansion

### 4.2.1 No expansion of function macro names not followed by parenthesis

ID	Status	Sources
[no-expand-function-non-function]	TBD	[C§6.10 Preprocessing directives] [G14§3.3 Macro Arguments]

String constants are output without being examined for macro expansion.

### 4.2.2 Function macro invocation may cross logical line boundaries

ID	Status	Sources
[expand-function-macro-multiline]	TBD	[C§6.10 Preprocessing directives] [G§3.3 Macro Arguments] [G14§3.10.5 Self-Referential Macros]

String constants are output without being examined for macro expansion.

### 4.2.3 No expansion of self-referential macro names

ID	Status	Sources
[no-expand-self-reference]	TBD	[C§6.10 Preprocessing directives] [G14§3.10.5 Self-Referential Macros]

String constants are output without being examined for macro expansion.

### 4.2.4 No expansion in strings

ID	Status	Sources
[no-expand-string]	TBD	ble1, [Flpp§] [C§6.10 Preprocessing directives]

String constants are output without being examined for macro expansion.

#### 4.2.5 No expansion in Hollerith

ID	Status	Sources
[no-expand-hollerith]	TBD	ble1 [C§6.10 Preprocessing directives]

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

#### 4.2.6 No expansion in IMPLICIT single-character specifiers

ID	Status	Sources
[no-expand-implicit-char-list]	TBD	ble1

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

Note that this implies the preprocessor recognizes IMPLICIT statements.

#### 4.2.7 No expansion in FORMAT specifiers

ID	Status	Sources
[no-expand-format]	TBD	ble1, [Flpp§] [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.

#### 4.2.8 Expand INCLUDE lines as if #include

ID	Status	Sources
[preprocess-fortran-include]	TBD	[Flpp§], jor1, JoR4

Assuming the preprocessor is a mandatory part of the Fortran standard, preprocessor directives are allowed in the file specified in a Fortran INCLUDE line. Therefore, the preprocessor should process the INCLUDE-ed file as if it had been invoked via the #include directive.

Otherwise, where will the handling of directives the included file be handled, and how can it use any of the macro definitions available at the time the INCLUDE statement is encountered. (It is likely to be included in multiple places in the application.)

#### 4.2.9 Expand macro names in *kind-param* in literal constants

ID	Status	Sources
[expand-kind-param]	TBD	JoR3 [F2023§7.4.3.1R709 Integer type] [F2023§7.4.3.2R714 Real type] [F2023§7.4.4.3R724 Character literal constant], [F2023§7.4.5R725 Logical type]

If the *kind-param* is a *scalar-int-constant-name* following the underscore in an *int-literal-constant*, *real-literal-constant*, and *logical-literal-constant*, that constant name is subject to macro expansion. This needs to be explicit, as otherwise the preprocessor might treat `_kind-name` as an identifier, as many preprocessor predefined macro names begin with an underscore.

In a *char-literal-constant*, if the *kind-param* preceding the underscore (`_`) is a *scalar-int-constant-name*, that constant name is subject to macro expansion. This needs to be explicit, as otherwise the preprocessor might treat `kind-name_` as an identifier.

## 5 Syntax

The EBNF grammar recognized in Phase 4.

```

/*
 * A Bison grammar for Phase 4 of the Fortran preprocessor.
 *
 * This grammar assumes the tokenization of the input stream
 * performed in Phase 3. As such, we don't see whitespace
 * or comments. We do see identifiers, whole and real numbers,
 * and tokens that carry additional information.
 *
 * In general, the grammar rules follow Clause 6.10 of
 * the C programming language standard (ISO/IEC 9899:2018, IDT).
 *
 * The grammar rules for expressions represent the Fortran
 * standard's expression rules in clause 10.1.2.
 */

%token          HASH_DEFINE "#define"
%token          HASH_ELIF  "#elif"
%token          HASH_ELSE  "#else"
%token          HASH_ENDIF "#endif"
%token          HASH_ERROR "#error"
%token          HASH_IF    "#if"
%token          HASH_IFDEF "#ifdef"
%token          HASH_IFNDEF "#ifndef"
%token          HASH_INCLUDE "#include"
%token          HASH_LINE  "#line"
%token          HASH_PRAGMA "#pragma"
%token          HASH_UNDEF "#undef"
%token          HASH_WARNING "#warning"

%left          AMPERSAND "&"
%left          AMPERSAND_AMPERSAND "&&"
%token          AT "@"

```



%token	BANG "!"	
%left	BANG_EQ "!="	
%left	BAR " "	
%left	BAR_BAR "  "	
%left	CARET "^"	
%token	COLON ":"	
%token	COLON_COLON "::"	
%token	COMMA ","	
%token	DOLLAR "\$"	
%token	ELLIPSES "..."	
%token	EO_ARGS	
%token	EOL	
%left	EQ "="	
%left	EQ_EQ "=="	
%token	FORMAT "format"	
%left	GT ">"	
%left	GT_EQ ">="	
%left	GT_GT ">>"	
%left	HASH "#"	
%left	HASH_HASH "##"	
%token	ID	
%token	ID_LPAREN	/* only on #define */
%token	IMPLICIT "implicit"	
%token	LBRACKET "["	
%token	LPAREN "("	
%token	LPAREN_SLASH "(/"	
%left	LT "<"	
%left	LT_EQ "<="	
%left	LT_LT "<<"	
%left	MINUS "-"	
%left	PERCENT "%"	
%token	PERIOD "."	
%left	PERIOD_AND_PERIOD ".and."	
%left	PERIOD_EQ_PERIOD ".eq."	
%left	PERIOD_EQV_PERIOD ".eqv."	
%token	PERIOD_FALSE_PERIOD ".false."	
%left	PERIOD_GE_PERIOD ".ge."	
%left	PERIOD_GT_PERIOD ".gt."	
%left	PERIOD_ID_PERIOD	/* user-defined operator */
%left	PERIOD_LE_PERIOD ".le."	
%left	PERIOD_LT_PERIOD ".lt."	
%left	PERIOD_NE_PERIOD ".ne."	
%left	PERIOD_NEQV_PERIOD ".neqv."	
%token	PERIOD_NIL_PERIOD ".nil."	
%left	PERIOD_NOT_PERIOD ".not."	
%left	PERIOD_OR_PERIOD ".or."	
%token	PERIOD_TRUE_PERIOD ".true."	
%left	PLUS "+"	
%left	POINTS "=>"	
%token	QUESTION "?"	
%token	RBRACKET "]"	
%token	REAL_NUMBER	
%token	RPAREN ")"	
%left	SEMICOLON ";"	
%left	SLASH "/"	
%left	SLASH_EQ "/="	
%token	SLASH_RPAREN ")/"	
%left	SLASH_SLASH "//"	
%token	STRING	
%token	TILDE "~"	
%left	TIMES "**"	
%right	TIMES_TIMES "***"	
%token	UNDERSCORE "_"	/* for _KIND, not ID */
%token	WHOLE_NUMBER	
%token	UND_UND_FILE "--FILE--"	
%token	UND_UND_LINE "--LINE--"	
%token	UND_UND_DATE "--DATE--"	
%token	UND_UND_TIME "--TIME--"	
%token	UND_UND_STDFORTRAN "STDFORTRAN"	
%token	UND_UND_STDFORTRAN_VERSION "__STDFORTRAN_VERSION__"	
%token	UND_UND_VA_ARGS "VA_ARGS"	
%token	UND_UND_VA_OPT "VA_OPT"	
%%		

```

ExecutableProgram:
    CommandLineDefinitionList EO_ARGS PreprocessingFile
;

CommandLineDefinitionList:
    /* empty */
    | CommandLineDefinitionList CommandLineDefinition
;

CommandLineDefinition:
    IncludePath EOL
    | DefineArgument EOL
    | UndefineArgument EOL
;

IncludePath:
    HASH_INCLUDE STRING
;

DefineArgument:
    HASH_DEFINE ID ReplacementText
    | HASH_DEFINE ID_LPAREN LambdaList RPAREN ReplacementText
;

UndefineArgument:
    HASH_UNDEF ID
;

PreprocessingFile:
    /* empty */
    | GroupPartList
;

/* A GroupPart is some directive, or some Fortran text. */
GroupPartList:
    GroupPart
    | GroupPartList GroupPart
;

GroupPart:
    IfSection
    | ControlLine
    | NonDirective
    | FortranSourceLine
;

IfSection:
    HASH_IF Expression EOL HASH_ENDIF EOL
    | HASH_IF Expression EOL ElseGroup HASH_ENDIF EOL
    | HASH_IF Expression EOL ElifGroupList HASH_ENDIF EOL
    | HASH_IF Expression EOL ElifGroupList ElseGroup HASH_ENDIF EOL
    | HASH_IFDEF ID EOL GroupPartList HASH_ENDIF EOL
    | HASH_IFNDEF ID EOL GroupPartList HASH_ENDIF EOL
;

ElifGroupList:
    HASH_ELIF ID EOL GroupPartList
    | ElifGroupList HASH_ELIF ID EOL GroupPartList
;

ElseGroup:
    HASH_ELSE EOL GroupPartList
;

ControlLine:
    IncludeControlLine
    | DefineIdControlLine
    | DefineFunctionControlLine
    | LineControlLine
    | ErrorControlLine
    | WarningControlLine
    | PragmaControlLine
;

```

```

/* TODO Add PPTokens as alternative. */
IncludeControlLine:
    HASH_INCLUDE STRING EOL
    ;

DefineIdControlLine:
    HASH_DEFINE ID EOL
    | HASH_DEFINE ID PPTokenList EOL
    ;

/*
 * Parameter lists on macro functions are comma-separated
 * identifiers.
 */
DefineFunctionControlLine:
    HASH_DEFINE ID_LPAREN LambdaList RPAREN EOL
    | HASH_DEFINE ID_LPAREN LambdaList RPAREN ReplacementText EOL
    ;

LambdaList:
    /* empty */
    | ELLIPSES
    | IDList
    | IDList COMMA ELLIPSES
    ;

IDList:
    ID
    | IDList COMMA ID
    ;

LineControlLine:
    HASH_LINE STRING WHOLE_NUMBER EOL
    | HASH_LINE WHOLE_NUMBER EOL
    ;

ErrorControlLine:
    HASH_ERROR STRING EOL
    ;

WarningControlLine:
    HASH_WARNING STRING EOL
    ;

PragmaControlLine:
    HASH_PRAGMA PPTokenList EOL
    ;

NonDirective:
    HASH PPTokenList EOL
    ;

ReplacementText:
    ReplacementToken
    | ReplacementText ReplacementToken
    ;

/*
 * '#' and '##' operators can only appear in the replacement
 * text in #define directives.
 */
ReplacementToken:
    PPToken
    | HASH
    | HASH_HASH
    ;

PPTokenList:
    PPToken
    | PPTokenList PPToken
    ;

PPToken:
    FortranToken
    | CPPToken
    ;

```

```

PPTokenListExceptCommaRParen :
    PPTokenExceptCommaRParen
    | PPTokenListExceptCommaRParen PPTokenExceptCommaRParen
    ;

PPTokenExceptCommaRParen :
    FortranTokenExceptCommaRParen
    | CPPToken
    ;

/*
 * This should include every token that the tokenizer
 * could recognize. The tokenizer has to do some recognition
 * of Fortran operators (such as .AND.) and places where
 * preprocessing expansion should not * occur (such as FORMAT
 * and IMPLICIT).
 */

FortranTokenList :
    FortranToken
    | FortranTokenList FortranToken
    ;

FortranToken :
    FortranTokenAnywhere
    | COMMA
    | RPAREN
    | FORMAT
    | IMPLICIT
    ;

FortranTokenExceptCommaRParen :
    FortranTokenAnywhere
    | FORMAT
    | IMPLICIT
    ;

FortranTokenExceptFormatExplicit :
    FortranTokenAnywhere
    | COMMA
    | RPAREN
    ;

FortranTokenAnywhere :
    AT
    | COLON
    | COLON_COLON
    | DOLLAR
    | EQ
    | EQ_EQ
    | GT
    | GT_EQ
    | ID_
    | LBRACKET
    | LPAREN
    | LT
    | LT_EQ
    | MINUS
    | PERCENT
    | PERIOD
    | PERIOD_AND_PERIOD
    | PERIOD_EQ_PERIOD
    | PERIOD_EQV_PERIOD
    | PERIOD_FALSE_PERIOD
    | PERIOD_GE_PERIOD
    | PERIOD_GT_PERIOD
    | PERIOD_ID_PERIOD
    | PERIOD_LE_PERIOD
    | PERIOD_LT_PERIOD
    | PERIOD_NE_PERIOD
    | PERIOD_NEQV_PERIOD
    | PERIOD_NIL_PERIOD
    | PERIOD_NOT_PERIOD
    | PERIOD_OR_PERIOD
    | PERIOD_TRUE_PERIOD
    | /* user-defined operator */

```

```

| PLUS
| POINTS
| QUESTION
| RBRACKET
| REAL_NUMBER
| SEMICOLON
| SLASH
| SLASH_EQ
| SLASH_SLASH
| STRING
| TIMES
| TIMES_TIMES
| UNDERSCORE
| WHOLE_NUMBER
|
;

FortranTokenListExceptFormatExplicit:
FortranTokenExceptFormatExplicit
| FortranTokenListExceptFormatExplicit FortranTokenExceptFormatExplicit
;

/*
* Tokens that can appear in C preprocessor replacement text
* in addition to the Fortran tokens.
*/
CPPToken:
AMPERSAND
| AMPERSAND_AMPERSAND
| BANG
| BANG_EQ
| BAR
| BAR_BAR
| CARET
| GT_GT
| LT_LT
| TILDE
;

/* Following Fortran ISO/IEC 1539-1:2023 Clause 10.1.2
*
* modified for C-like syntax
*
*
* INCITS and WG5 have agreed (so far) that the preprocessor
* should conform to a subset of the C preprocessor
* expression syntax. There has been no consensus
* to include the standard Fortran operators, but
* we include them here for completeness. (It is easier
* to discuss removing them than adding them.)
*
* Note that operator precedence differs between C
* and Fortran. The grammar below attempts to merge
* these precedence lists, leaning towards C's
* operator precedence.
*/
Expression:
ConditionalExpr
| Expression EquivOp ConditionalExpr
;

EquivOp:
PERIOD_EQV_PERIOD
| PERIOD_NEQV_PERIOD
;

ConditionalExpr:
LogicalOrExpr QUESTION Expression COLON ConditionalExpr
| LogicalOrExpr
;

LogicalOrExpr:
LogicalAndExpr
| LogicalOrExpr OrOp LogicalAndExpr
;

OrOp:

```

```

        BAR_BAR
    | PERIOD_OR_PERIOD
    ;

LogicalAndExpr:
    InclusiveOrExpr
    | LogicalAndExpr AndOp InclusiveOrExpr
    ;

AndOp:
    AMPERSAND_AMPERSAND
    | PERIOD_AND_PERIOD
    ;

InclusiveOrExpr:
    ExclusiveOrExpr
    | InclusiveOrExpr BAR ExclusiveOrExpr
    ;

ExclusiveOrExpr:
    AndExpr
    | ExclusiveOrExpr CARET AndExpr
    ;

AndExpr:
    EqualityExpr
    | AndExpr AMPERSAND EqualityExpr
    ;

EqualityExpr:
    RelationalExpr
    | EqualityExpr EqualityOp RelationalExpr
    ;

EqualityOp:
    PERIOD_EQ_PERIOD
    | PERIOD_NE_PERIOD
    | EQ_EQ
    | SLASH_EQ
    | BANG_EQ
    ;

RelationalExpr:
    ShiftExpr
    | RelationalExpr RelationalOp ShiftExpr
    ;

RelationalOp:
    PERIOD_LE_PERIOD
    | PERIOD_LT_PERIOD
    | PERIOD_GE_PERIOD
    | PERIOD_GT_PERIOD
    | LT
    | GT
    | LT_EQ
    | GT_EQ
    ;

ShiftExpr:
    CharacterExpr
    | ShiftExpr ShiftOp CharacterExpr
    ;

ShiftOp:
    LT_LT
    | GT_GT
    ;

CharacterExpr:
    AdditiveExpr
    | CharacterExpr SLASH_SLASH AdditiveExpr
    ;

AdditiveExpr:
    MultiplicativeExpr
    | AdditiveExpr AddOp MultiplicativeExpr

```

```

;
AddOp:
    PLUS
    | MINUS
    ;
MultiplicativeExpr:
    PowerExpr
    | MultiplicativeExpr MultOp PowerExpr
    ;
MultOp:
    TIMES
    | SLASH
    | PERCENT
    ;
PowerExpr:
    UnaryExpr
    | UnaryExpr TIMES_TIMES PowerExpr
    ;
UnaryExpr:
    UnaryOp PostfixExpr
    | PostfixExpr
    ;
UnaryOp:
    PLUS
    | MINUS
    | PERIOD_NOT_PERIOD
    | BANG
    | TILDE
    ;
PostfixExpr:
    PrimaryExpr
    | ID LPAREN RPAREN
    | ID LPAREN ActualArgumentList RPAREN
    ;
/* TODO: Really this should be properly nested parenthesized lists */
ActualArgumentList:
    PPTokenListExceptCommaRParen
    | ActualArgumentList COMMA PPTokenListExceptCommaRParen
    ;
/* Real numbers aren't allowed in conditional expressions */
PrimaryExpr:
    WHOLE_NUMBER
    | ID
    | PERIOD_FALSE_PERIOD
    | PERIOD_NIL_PERIOD
    | PERIOD_TRUE_PERIOD
    | LPAREN Expression RPAREN
    | PredefinedIdentifier
    ;
/* Identifiers known to the preprocessor (such as __FILE__) */
PredefinedIdentifier:
    UND_UND_FILE
    | UND_UND_LINE
    | UND_UND_DATE
    | UND_UND_TIME
    | UND_UND_STDFORTRAN
    | UND_UND_STDFORTRAN_VERSION
    | UND_UND_VA_ARGS
    | UND_UND_VA_OPT
    /* | ProcessorDefinedPPIIdentifier */
    ;
/* /\* Implementation-defined predefined identifiers *\// */
/* ProcessorDefinedPPIIdentifier: */
/* ; */

```

```
FortranSourceLine :  
    EOL  
    | FORMAT FortranTokenList EOL  
    | IMPLICIT FortranTokenList EOL  
    | FortranTokenListExceptFormatExplicit EOL  
    ;
```

## References

- [ISO18] ISO/IEC. Information technology – programming languages – C. Standard ISO/IEC 9899:2018, International Organization for Standardization, Geneva, CH, July 2018.
- [ISO23] ISO/IEC JTC 1. Information technology – programming languages – Fortran. Standard ISO/IEC 1539-1:2023, International Organization for Standardization, Geneva, CH, November 2023.
- [ISO24] ISO/IEC. Information technology – programming languages – C. Standard ISO/IEC 9899:2018 (R2024), International Organization for Standardization, Geneva, CH, October 2024.
- [Sub24] INCITS/Fortran JoR Subgroup. Preprocessor, take 2, Feb 2024.
- [SW24] Richard M. Stallman and Zachary Weinberg. The C preprocessor, 2024.