

---

# **Fparser Reference Guide**

***Release 0.0.16***

**R. W. Ford, P. Peterson, A. R. Porter**

**Oct 13, 2022**



# CONTENTS

<b>1</b>	<b>Sphinx and AutoAPI-generated Documentation</b>	<b>3</b>
1.1	fparser . . . . .	3
<b>2</b>	<b>Doxygen-generated Documentation</b>	<b>117</b>
	<b>Python Module Index</b>	<b>119</b>
	<b>Index</b>	<b>121</b>



Welcome to the fparser Reference Guide. This consists of auto-generated API documentation produced using both the *AutoAPI Sphinx plugin* and *Doxygen*.



## SPHINX AND AUTOAPI-GENERATED DOCUMENTATION

### 1.1 `fparser`

- *Submodules*

#### 1.1.1 Submodules

##### `fparser.api`

Public API for Fortran parser.

##### Module content

- *Module content*

##### `fparser.common`

- *Submodules*

##### Submodules

##### `fparser.common.base_classes`

Base classes for all Fortran statement types

- *Classes*

## Classes

- *Statement*: Statement instance has attributes:
- *BeginStatement*: [ construct\_name : ] <blocktype> [ <name> ]
- *EndStatement*: END [<blocktype> [<name>]]
- *Variable*: Variable instance has attributes:
- *AttributeHolder*: Defines a object with predefined attributes. Only those attributes
- *ProgramBlock*: Undocumented.

**class** fparser.common.base\_classes.**Statement**(parent, item)

**Statement instance has attributes:**

parent - Parent BeginStatement or FortranParser instance  
item - Line instance containing the statement line  
isvalid - boolean, when False, the Statement instance will be ignored

## Inheritance

Statement

**get\_provides()**

Returns dictionary containing statements that block provides or None when N/A.

**get\_type(name)**

Return type declaration using implicit rules for name.

**get\_variable(name)**

Return Variable instance of variable name.

**class** fparser.common.base\_classes.**BeginStatement**(parent, item=None)

[ construct\_name : ] <blocktype> [ <name> ]

**BeginStatement instances have additional attributes:**

name blocktype

**Block instance has attributes:**

content - list of Line or Statement instances  
name - name of the block, unnamed blocks are named with the line label

construct\_name - name of a construct  
parent - Block or FortranParser instance  
item - Line instance containing the block start statement  
get\_item, put\_item - methods to retrieve/submit Line instances

from/to Fortran reader.

isvalid - boolean, when False, the Block instance will be ignored.

stmt\_cls, end\_stmt\_cls



## Inheritance



**fill**(*end\_flag=False*)

Fills blocks content until the end of block statement.

**handle\_unknown\_item\_and\_raise**(*item*)

Called when process\_subitem does not find a start or end of block. It adds the item (which is an instance of Line) to the content, but then raises an AnalyzeError. An instance of Line in content typically results in other errors later (e.g. because Line has no analyze method).

**process\_item**()

Process the line

**process\_subitem**(*item*)

Check if item is blocks start statement, if it is, read the block.

Return True to stop adding items to given block.

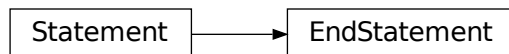
**class** fparser.common.base\_classes.**EndStatement**(*parent, item*)

END [<blocktype> [<name>]]

**EndStatement instances have additional attributes:**

name blocktype

## Inheritance



**tofortran**(*isfix=None*)

Returns a valid Fortran string for this END statement. It guarantees that there is no white space after the 'END' in case of an unnamed statement.

### Parameters

**isfix** (*bool*) – True if the code is in fixed format.

### Returns

the (named or unnamed) valid Fortran END statement as a string.

### Return type

str

**class** fparser.common.base\_classes.**Variable**(*parent, name*)

**Variable instance has attributes:**

name typedcl dimension attributes intent parent - Statement instances defining the variable

### Inheritance

Variable

**class** fparser.common.base\_classes.**AttributeHolder**(\*\**kws*)

Defines a object with predefined attributes. Only those attributes are allowed that are specified as keyword arguments of a constructor. When an argument is callable then the corresponding attribute will be read-only and set by the value the callable object returns.

### Inheritance

AttributeHolder

**class** fparser.common.base\_classes.**ProgramBlock**

### Inheritance

ProgramBlock

**fparser.common.readfortran**

Provides Fortran reader classes.

## Overview

Provides FortranReader classes for reading Fortran codes from files and strings. FortranReader handles comments and line continuations of both fix and free format Fortran codes.

## Examples

```
>>> from fparser.common.readfortran import FortranFileReader
>>> import os
>>> reader = FortranFileReader(os.path.expanduser('~/.src/blas/daxpy.f'))
>>> print reader.next()
line #1 'subroutine daxpy(n,da,dx,incx,dy,incy)'
>>> print `reader.next()`
Comment('c      constant times a vector plus a vector.\n
c      uses unrolled loops for increments equal to one.\n
c      jack dongarra, linpack, 3/11/78.\n
c      modified 12/3/93, array(1) declarations changed to array(*)',(3, 6))
>>> print `reader.next()`
Line('double precision dx(*),dy(*),da',(8, 8),'')
>>> print `reader.next()`
Line('integer i,incx,incy,ix,iy,m,mp1,n',(9, 9),'')
```

Note that the `.next()` method may return *Line*, *SyntaxErrorLine*, *Comment*, *MultiLine*, or *SyntaxErrorMultiLine* instance. Let us continue with the above example session to illustrate the *Line* methods and attributes:

```
>>> item = reader.next()
>>> item
Line('if (da .eq. 0.0d0) return',(12, 12),'')
>>> item.line
'if (da .eq. 0.0d0) return'
>>> item.strline
'if (F2PY_EXPR_TUPLE_5) return'
>>> item.strlinemap
{'F2PY_EXPR_TUPLE_5': 'da .eq. 0.0d0'}
>>> item.span
(12, 12)
>>> item.get_line()
'if (F2PY_EXPR_TUPLE_5) return'
```

To read a Fortran code from a string, use *FortranStringReader* class:

```
>>> from fparser.common.sourceinfo import FortranFormat
>>> from fparser.common.readfortran import FortranStringReader
>>> code = '''
...     subroutine foo(a)
...         integer a
...         print*, "a=",a
...     end
... '''
>>> reader = FortranStringReader(code)
>>> reader.set_format(FortranFormat(False, True))
```

(continues on next page)

(continued from previous page)

```
>>> reader.next()
      Line('subroutine foo(a)',(2, 2),'')
>>> reader.next()
      Line('integer a',(3, 3),'')
>>> reader.next()
      Line('print*, "a=",a',(4, 4),'')
```

- *Classes*
- *Exceptions*

## Classes

- *FortranFileReader*: Constructs a FortranFileReader object from a file.
- *FortranStringReader*: Reads Fortran source code as a string.
- *Line*: Holds a Fortran source line.
- *Comment*: Holds a Fortran comment.
- *MultiLine*: Holds PYF file multiline.

```
class fparser.common.readfortran.FortranFileReader(file_candidate, include_dirs=None,
                                                    source_only=None, ignore_comments=True)
```

Constructs a FortranFileReader object from a file.

### Parameters

- **file\_candidate** – A filename or file-like object.
- **include\_dirs** (*list*) – Directories in which to look for inclusions.
- **source\_only** (*list*) – Fortran source files to search for modules required by “use” statements.
- **ignore\_comments** (*bool*) – Whether or not to ignore comments

For example:

```
>>> from fparser.common.readfortran import FortranFileReader
>>> import os
>>> reader = FortranFileReader('myfile.f90')
```

## Inheritance



**close\_source()**

Called when `self.source.next()` raises `StopIteration`.

**class** `fparser.common.readfortran.FortranStringReader`(*string*, *include\_dirs=None*, *source\_only=None*, *ignore\_comments=True*)

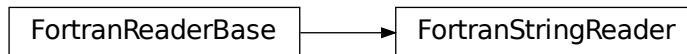
Reads Fortran source code as a string.

**Parameters**

- **string** (*str*) – string to read
- **include\_dirs** (*list*) – List of dirs to search for include files
- **source\_only** (*list*) – Fortran source files to search for modules required by “use” statements.
- **ignore\_comments** (*bool*) – Whether or not to ignore comments

For example:

```
>>> from fparser.common.readfortran import FortranStringReader
>>> code = '''
    subroutine foo(a)
        integer a
        print*, "a=", a
    end
    '''
>>> reader = FortranStringReader(code)
```

**Inheritance**

**class** `fparser.common.readfortran.Line`(*line*, *linenospan*, *label*, *name*, *reader*)

Holds a Fortran source line.

Attributes

**line**

[*str*] code line

**span**

[2-tuple] starting and ending line numbers

**label**

[{*int*, *None*}] Specify statement label

**name**

[{*str*, *None*}] Specify construct name.

*reader* : `FortranReaderBase` *strline* : {*None*, *str*} *is\_f2py\_directive* : *bool*

the line contains f2py directive

## Inheritance

Line

### **apply\_map**(*line*)

Substitutes magic strings in a line with values specified in a map.

### **clone**(*line*)

This Line has its contents overwritten by the passed string. The incoming string has substitution applied.

### **copy**(*line=None, apply\_map=False*)

Creates a Line object from a string.

If no line argument is specified a copy is made of this Line.

If a substitution map is provided it is used while making the copy.

### **has\_map**()

Returns true when a substitution map has been registered.

### **class** fparser.common.readfortran.**Comment**(*comment, linenospan, reader*)

Holds a Fortran comment.

#### Parameters

- **comment** (*str*) – String containing the text of a single or multi-line comment
- **linenospan** (*(int, int)*) – A 2-tuple containing the start and end line numbers of the comment from the input source.
- **reader** (*fparser.common.readfortran.FortranReaderBase*) – The reader object being used to read the input source.

## Inheritance

Comment

### **isempty**(*ignore\_comments=False*)

Whether or not this comment is in fact empty (or we are ignoring it). Provided for compatibility with Line.isempty()

**Parameters**

**ignore\_comments** (*bool*) – whether we ignore comments

**Returns**

True if we are ignoring comments, False otherwise

**Return type**

bool

**class** `fparser.common.readfortran.MultiLine`(*prefix, block, suffix, linenospan, reader*)

Holds PYF file multiline.

**PYF file multiline is represented as follows::**

`prefix+''' +lines+''' +suffix.`

**Parameters**

- **prefix** (*str*) – the prefix of the line(s)
- **block** (List[`fparser.common.readfortran.Line`]) – list of lines
- **suffix** (*str*) – the suffix of the block of lines
- **linenospan** (*Tuple[int, int]*) – starting and ending line numbers
- **reader** (`fparser.common.readfortran.FortranReaderBase`) – the current reader instance.

**Inheritance**

MultiLine

**isempty**(*ignore\_comments=False*)

Returns true if there is no significant text in this multi-line string.

**Exceptions**

- `FortranReaderError`: Thrown when there is an error reading the Fortran source file.
- `SyntaxErrorLine`: Indicates a syntax error while processing a line.
- `SyntaxErrorMultiLine`: Indicates a syntax error while processing Python multi-line strings.

**exception** `fparser.common.readfortran.FortranReaderError`

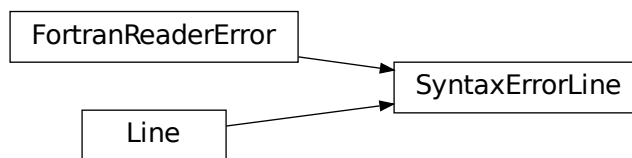
Thrown when there is an error reading the Fortran source file.

## Inheritance

FortranReaderError

**exception** `fparser.common.readfortran.SyntaxErrorLine`(*line, linenospan, label, name, reader, message*)  
Indicates a syntax error while processing a line.

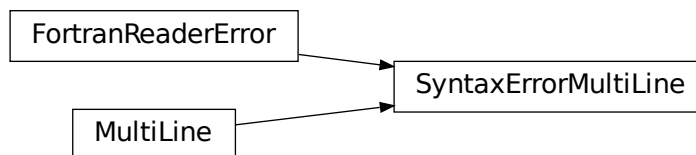
## Inheritance



**exception** `fparser.common.readfortran.SyntaxErrorMultiLine`(*prefix, block, suffix, linenospan, reader, message*)

Indicates a syntax error while processing Python multi-line strings.

## Inheritance





**fparser.common.sourceinfo**

Provides functions to determine whether a piece of Fortran source is free or fixed format. It also tries to differentiate between strict and “pyf” although I’m not sure what that is.

**fparser.common.splitline**

Defines LineSplitter and helper functions.

Original Author: Pearu Peterson <pearu@cens.ioc.ee> First version created: May 2006

- *Functions*
- *Classes*

**Functions**

- *string\_replace\_map()*: Undocumented.
- *splitquote()*: Fast LineSplitter
- *splitparen()*: Splits a line into top-level parenthesis and not-parenthesised

`fparser.common.splitline.string_replace_map(*args, **kwargs)`

`fparser.common.splitline.splitquote(line, stopchar=None, lower=False, quotechars="\\"")`

Fast LineSplitter

`fparser.common.splitline.splitparen(line, paren_open='(', paren_close=')')`

Splits a line into top-level parenthesis and not-parenthesised parts. E.g.: “a( (1+2)\*3) = b(x)” becomes: [“a”, “( (1+2)\*3)”, “ = b”, “(x)”]. :param str line: the string to split. :param str paren\_open: The characters that define an open parentheses. :param str paren\_close: The characters that define a closing parentheses. :return: List of parenthesised and not-parenthesised parts :rtype: list of str The paren\_open and paren\_close strings must be matched in order: paren\_open[x] is closed by paren\_close[x].

**Classes**

- *String*: Dummy string class.

`class fparser.common.splitline.String`

Dummy string class.

## Inheritance

String

`fparser.common.tests`

- *Submodules*

## Submodules

`fparser.common.tests.logging_utils`

Helps with testing methods which write to the standard logger.

`fparser.common.utils`

Various utility functions.

Permission to use, modify, and distribute this software is given under the terms of the NumPy License. See <http://scipy.org>.

NO WARRANTY IS EXPRESSED OR IMPLIED. USE AT YOUR OWN RISK. Author: Pearu Peterson  
<pearu@cens.ioc.ee> Created: May 2006

- *Functions*
- *Classes*
- *Exceptions*
- *Variables*

## Functions

- `split_comma()`: Split (an optionally bracketed) comma-separated list into
- `specs_split_comma()`: Undocumented.
- `get_module_file()`: Undocumented.
- `parse_bind()`: Undocumented.
- `parse_result()`: Undocumented.

- `parse_array_spec()`: Undocumented.
- `str2stmt()`: Convert Fortran code to Statement tree.

`fparser.common.utils.split_comma(line, item=None, comma=',', keep_empty=False, brackets=None)`

Split (an optionally bracketed) comma-separated list into items and return a list containing them. If supplied then brackets must be a list of containing two strings, the first being the opening bracket and the second the closing bracket.

`fparser.common.utils.specs_split_comma(line, item=None, upper=False)`

`fparser.common.utils.get_module_file(name, directory, _cache={})`

`fparser.common.utils.parse_bind(line, item=None)`

`fparser.common.utils.parse_result(line, item=None)`

`fparser.common.utils.parse_array_spec(line, item=None)`

`fparser.common.utils.str2stmt(string, isfree=True, isstrict=False)`

Convert Fortran code to Statement tree.

## Classes

- `classes`: Make classes available as attributes of this class.

`class fparser.common.utils.classes(name, bases, dict)`

Make classes available as attributes of this class.

To add a class to the attributes list, one must use:

```
class Name(metaclass=classes):
```

in the definition of the class.

In addition, apply the following tasks:

- decorate analyze methods with `show_item_on_failure`

## Inheritance

classes

## Exceptions

- *ParseError*: Common base class for all non-exit exceptions.
- *AnalyzeError*: Common base class for all non-exit exceptions.

**exception** `fparser.common.utils.ParseError`

### Inheritance

ParseError

**exception** `fparser.common.utils.AnalyzeError`

### Inheritance

AnalyzeError

## Variables

- *is\_name*

`fparser.common.utils.is_name`

Matches zero or more characters at the beginning of the string.

```
<built-in method match of re.Pattern object at 0x5582489511b0>
```

**fparser.one**

- *Submodules*

**Submodules****fparser.one.block\_statements**

Fortran block statements.

- *Classes*
- *Variables*

**Classes**

- *BeginSource*: Fortran source content.
- *Module*: MODULE <name>
- *PythonModule*: PYTHON MODULE <name>
- *Program*: PROGRAM [name]
- *BlockData*: BLOCK DATA [<block-data-name>]
- *Interface*: INTERFACE [<generic-spec>] | ABSTRACT INTERFACE
- *Subroutine*: [<prefix>] SUBROUTINE <name> [( [<dummy-arg-list>] )]
- *Function*: [<prefix>] FUNCTION <name> ( [<dummy-arg-list>] ) [<suffix>]
- *SelectCase*: [<case-construct-name> :] SELECT CASE ( <case-expr> )
- *SelectType*: [<case-construct-name> :] SELECT TYPE ( <case-expr> )
- *WhereConstruct*: [<where-construct-name> :] WHERE ( <mask-expr> )
- *ForallConstruct*: [<forall-construct-name> :] FORALL <forall-header>
- *IfThen*: [<if-construct-name> :] IF ( <scalar-logical-expr> ) THEN
- *If*: IF ( <scalar-logical-expr> ) action-stmt
- *Do*: [<do-construct-name> :] DO label [loopcontrol]
- *Associate*: [<associate-construct-name> :] ASSOCIATE ( <association-list> )
- *TypeDecl*: TYPE [[, <typ-attr-spec-list>] ::] <type-name> [( <type-param-name-list> )]
- *Enum*: ENUM , BIND(C)
- *EndSource*: Dummy End statement for BeginSource.
- *EndModule*: END [<blocktype> [<name>]]
- *EndPythonModule*: END [<blocktype> [<name>]]
- *EndProgram*: END [PROGRAM [name]]

- *EndBlockData*: END [BLOCK DATA [<block-data-name>]]
- *EndInterface*: END [<blocktype> [<name>]]
- *EndSubroutine*: END [SUBROUTINE [name]]
- *EndFunction*: END [FUNCTION [name]]
- *EndSelect*: END [<blocktype> [<name>]]
- *EndWhere*: END WHERE [<where-construct-name>]
- *EndForall*: END FORALL [<forall-construct-name>]
- *EndIfThen*: END IF [<if-construct-name>]
- *EndDo*: END DO [<do-construct-name>]
- *EndAssociate*: END ASSOCIATE [<associate-construct-name>]
- *EndType*: END TYPE [<type-name>]
- *EndEnum*: END ENUM
- *GeneralAssignment*: <variable> = <expr>
- *Assignment*: <variable> = <expr>
- *PointerAssignment*: <variable> = <expr>
- *Assign*: ASSIGN <label> TO <int-variable-name>
- *Call*: Call statement class
- *Goto*: GO TO <label>
- *ComputedGoto*: GO TO ( <label-list> ) [ , ] <scalar-int-expr>
- *AssignedGoto*: GO TO <int-variable-name> [ ( <label> [ , <label> ]... ) ]
- *Continue*: CONTINUE
- *Return*: RETURN [ <scalar-int-expr> ]
- *Stop*: STOP [ <stop-code> ]
- *Print*: PRINT <format> [ , <output-item-list>]
- *Read*: Read0: READ ( <io-control-spec-list> ) [ <input-item-list> ]
- *Read0*: Read0: READ ( <io-control-spec-list> ) [ <input-item-list> ]
- *Read1*: Read0: READ ( <io-control-spec-list> ) [ <input-item-list> ]
- *Write*: WRITE ( <io-control-spec-list> ) [<output-item-list>]
- *Flush*: FLUSH <file-unit-number>
- *Wait*: WAIT ( <wait-spec-list> )
- *Contains*: CONTAINS
- *Allocate*: ALLOCATE ( [ <type-spec> :: ] <allocation-list> [ , <alloc-opt-list> ] )
- *Deallocate*: DEALLOCATE ( <allocate-object-list> [ , <dealloc-opt-list> ] )
- *ModuleProcedure*: [ MODULE ] PROCEDURE [::] <procedure-name-list>
- *Access*: <access-spec> [ [::] <access-id-list> ]
- *Public*: <access-spec> [ [::] <access-id-list> ]

- *Private*: <access-spec> [ [::] <access-id-list> ]
- *Close*: CLOSE ( <close-spec-list> )
- *Cycle*: CYCLE [ <do-construct-name> ]
- *Backspace*: REWIND <file-unit-number>
- *Endfile*: REWIND <file-unit-number>
- *Rewind*: REWIND <file-unit-number>
- *Open*: OPEN ( <connect-spec-list> )
- *Format*: FORMAT <format-specification>
- *Save*: SAVE [ [::] <savd-entity-list> ]
- *Data*: DATA <data-stmt-set> [ [ , ] <data-stmt-set> ]...
- *Nullify*: NULLIFY ( <pointer-object-list> )
- *Use*: Parses USE statement.
- *Exit*: EXIT [ <do-construct-name> ]
- *Parameter*: PARAMETER ( <named-constant-def-list> )
- *Equivalence*: EQUIVALENCE <equivalence-set-list>
- *Dimension*: DIMENSION [ :: ] <array-name> ( <array-spec> )
- *Target*: TARGET [ :: ] <object-name> ( <array-spec> )
- *Pointer*: POINTER [ :: ] <pointer-decl-list>
- *Protected*: PROTECTED [ :: ] <entity-name-list>
- *Volatile*: VOLATILE [ :: ] <object-name-list>
- *Value*: VALUE [ :: ] <dummy-arg-name-list>
- *ArithmeticIf*: IF ( <scalar-numeric-expr> ) <label> , <label> , <label>
- *Intrinsic*: INTRINSIC [ :: ] <intrinsic-procedure-name-list>
- *Inquire*: INQUIRE ( <inquire-spec-list> )
- *Sequence*: SEQUENCE
- *External*: EXTERNAL [ :: ] <external-name-list>
- *Namelist*: NAMELIST / <namelist-group-name> / <namelist-group-object-list> [ [ , ]
- *Common*: COMMON [ / [ <common-block-name> ] / ] <common-block-object-list> [ [ , ] / [ <common-block-name> ] / <common-block-object-list> ]...
- *Optional*: OPTIONAL [ :: ] <dummy-arg-name-list>
- *Intent*: INTENT ( <intent-spec> ) [ :: ] <dummy-arg-name-list>
- *Entry*: ENTRY <entry-name> [ ( [ <dummy-arg-list> ] ) [ <suffix> ] ]
- *Import*: IMPORT [ [ :: ] <import-name-list> ]
- *ForallStmt*: FORALL <forall-header> <forall-assignment-stmt>
- *SpecificBinding*: PROCEDURE [ ( <interface-name> ) ] [ [ , <binding-attr-list> ] :: ]
- *GenericBinding*: GENERIC [ , <access-spec> ] :: <generic-spec> => <binding-name-list>

- *FinalBinding*: FINAL [ :: ] <final-subroutine-name-list>
- *Allocatable*: ALLOCATABLE [ :: ] <object-name> [ ( <deferred-shape-spec-list> ) ]
- *Asynchronous*: ASYNCHRONOUS [ :: ] <object-name-list>
- *Bind*: <language-binding-spec> [ :: ] <bind-entity-list>
- *Else*: ELSE [ <if-construct-name> ]
- *ElseIf*: ELSE IF ( <scalar-logical-expr> ) THEN [ <if-construct-name> ]
- *Case*: CASE <case-selector> [ <case-construct-name> ]
- *TypeIs*: TYPE IS <type-selector> [ <case-construct-name> ]
- *ClassIs*: CLASS <class-selector>
- *WhereStmt*: WHERE ( <mask-expr> ) <where-assignment-stmt>
- *ElseWhere*: ELSE WHERE ( <mask-expr> ) [ <where-construct-name> ]
- *Enumerator*: ENUMERATOR [ :: ] <enumerator-list>
- *FortranName*: FORTRANNAME <name>
- *Threadsafe*: THREADSAFE
- *Depend*: DEPEND ( <name-list> ) [ :: ] <dummy-arg-name-list>
- *Check*: CHECK ( <c-int-scalar-expr> ) [ :: ] <name>
- *CallStatement*: CALLSTATEMENT <c-expr>
- *CallProtoArgument*: CALLPROTOARGUMENT <c-type-spec-list>
- *Pause*: PAUSE [ <char-literal-constant>[<int-literal-constant> ]
- *Comment*: Attributes
- *Integer*: <declaration-type-spec> [ [ , <attr-spec> ] :: ] <entity-decl-list>
- *Real*: <declaration-type-spec> [ [ , <attr-spec> ] :: ] <entity-decl-list>
- *DoublePrecision*: <declaration-type-spec> [ [ , <attr-spec> ] :: ] <entity-decl-list>
- *Complex*: <declaration-type-spec> [ [ , <attr-spec> ] :: ] <entity-decl-list>
- *DoubleComplex*: <declaration-type-spec> [ [ , <attr-spec> ] :: ] <entity-decl-list>
- *Character*: <declaration-type-spec> [ [ , <attr-spec> ] :: ] <entity-decl-list>
- *Logical*: <declaration-type-spec> [ [ , <attr-spec> ] :: ] <entity-decl-list>
- *Byte*: <declaration-type-spec> [ [ , <attr-spec> ] :: ] <entity-decl-list>
- *TypeStmt*: <declaration-type-spec> [ [ , <attr-spec> ] :: ] <entity-decl-list>
- *Class*: <declaration-type-spec> [ [ , <attr-spec> ] :: ] <entity-decl-list>
- *Implicit*: IMPLICIT <implicit-spec-list>

```
class fparser.one.block_statements.BeginSource(parent, item=None)
    Fortran source content.
```



## Inheritance



**end\_stmt\_cls**

alias of *EndSource*

**process\_item()**

Process the line

**process\_subitem**(*item*)

Check if item is blocks start statement, if it is, read the block.

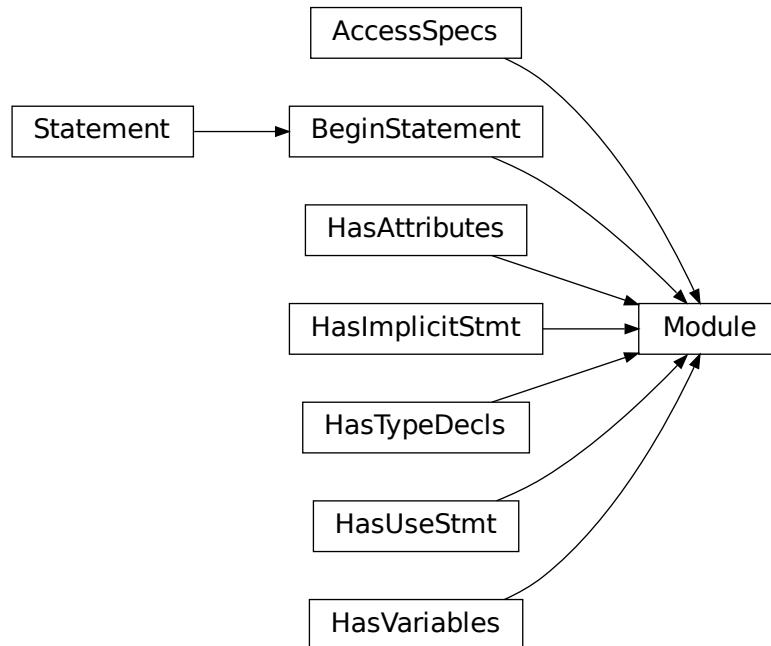
Return True to stop adding items to given block.

**class** fparser.one.block\_statements.**Module**(*parent*, *item=None*)

**MODULE** <name>

**END** [MODULE [name]]

## Inheritance



**end\_stmt\_cls**

alias of *EndModule*

**get\_provides()**

Returns dictionary containing statements that block provides or None when N/A.

**match**(pos=0, endpos=9223372036854775807)

Matches zero or more characters at the beginning of the string.

**process\_item()**

Process the line

**topyf**(tab="")

Constructs a pyf representation of this class.

**Parameters**

**tab** (str) – White space to prepend to output.

**Returns**

pyf code for this implicit statement.

**Return type**

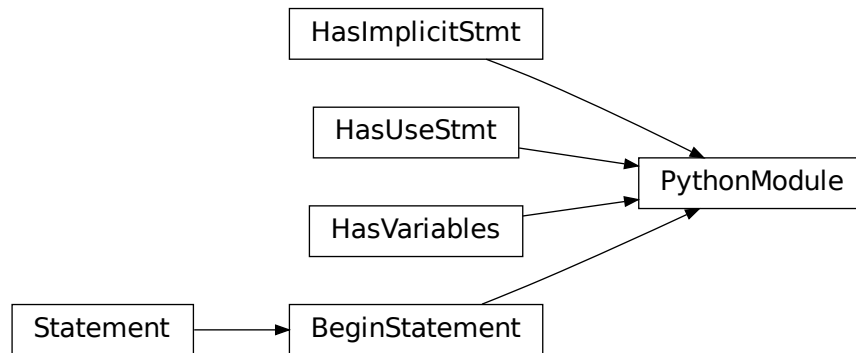
str

**class** fparser.one.block\_statements.**PythonModule**(parent, item=None)

**PYTHON MODULE** <name>

END [PYTHON MODULE [name]]

## Inheritance



**end\_stmt\_cls**

alias of *EndPythonModule*

**match**(*pos*=0, *endpos*=9223372036854775807)

Matches zero or more characters at the beginning of the string.

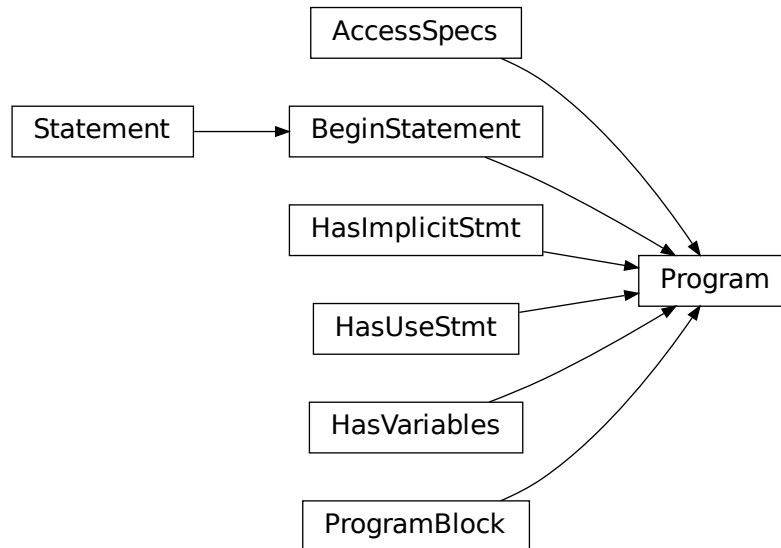
**process\_item**()

Process the line

**class** fparser.one.block\_statements.**Program**(*parent*, *item*=None)

PROGRAM [name]

## Inheritance



### **end\_stmt\_cls**

alias of *EndProgram*

**match**(*pos*=0, *endpos*=9223372036854775807)

Matches zero or more characters at the beginning of the string.

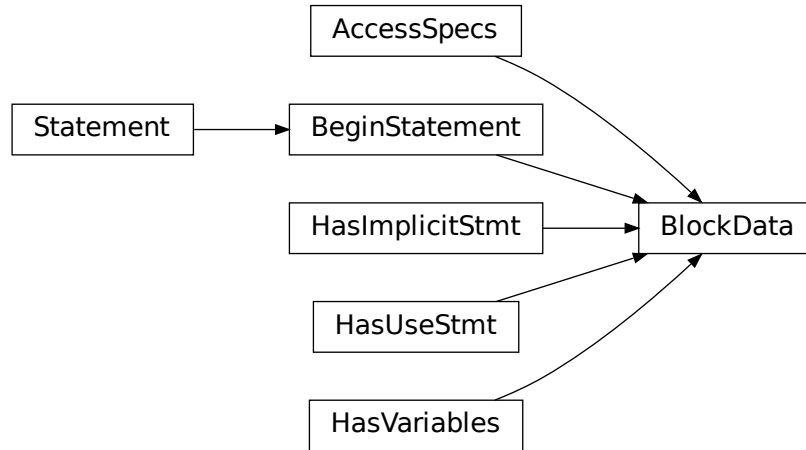
### **process\_item()**

Process the line

**class** fparser.one.block\_statements.**BlockData**(*parent*, *item*=None)

BLOCK DATA [<block-data-name>]

## Inheritance



### end\_stmt\_cls

alias of *EndBlockData*

**match**(pos=0, endpos=9223372036854775807)

Matches zero or more characters at the beginning of the string.

### process\_item()

Process the line

**class** fparser.one.block\_statements.**Interface**(parent, item=None)

INTERFACE [<generic-spec>] | ABSTRACT INTERFACE END INTERFACE [<generic-spec>]

<generic-spec> = <generic-name>

OPERATOR ( <defined-operator> )

ASSIGNMENT ( = )

<dtio-generic-spec>

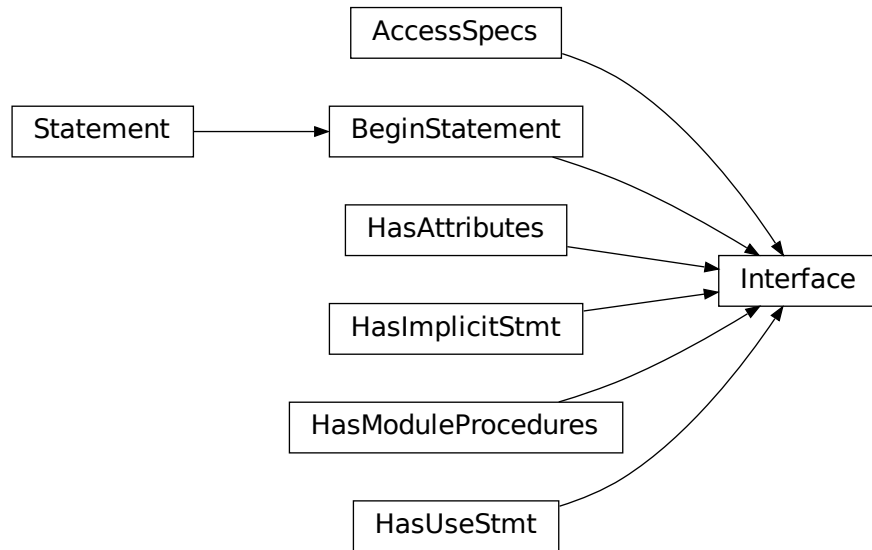
<dtio-generic-spec> = READ ( FORMATTED )

READ ( UNFORMATTED )

WRITE ( FORMATTED )

WRITE ( UNFORMATTED )

## Inheritance



**end\_stmt\_cls**

alias of *EndInterface*

**match**(*pos*=0, *endpos*=9223372036854775807)

Matches zero or more characters at the beginning of the string.

**process\_item**()

Process the line

**topyf**(*tab*="")

Constructs a pyf representation of this class.

**Parameters**

**tab** (*str*) – White space to prepend to output.

**Returns**

pyf code for this implicit statement.

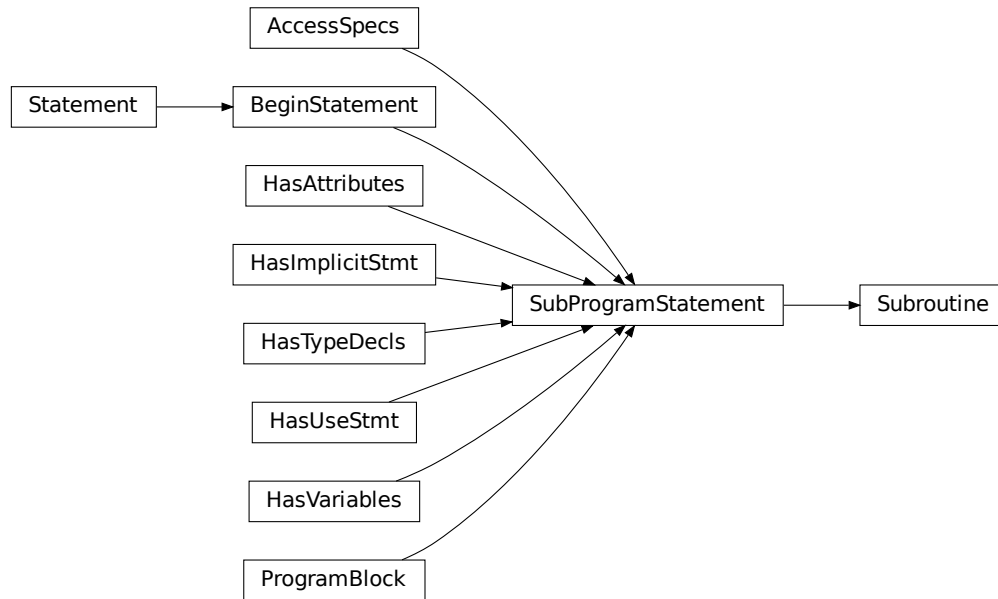
**Return type**

str

**class** fparser.one.block\_statements.**Subroutine**(*parent*, *item*=None)

[<prefix>] SUBROUTINE <name> [( [<dummy-arg-list>] ) [<proc-language-binding-spec>]]

## Inheritance



### end\_stmt\_cls

alias of *EndSubroutine*

**match**(pos=0, endpos=9223372036854775807)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Function**(parent, item=None)

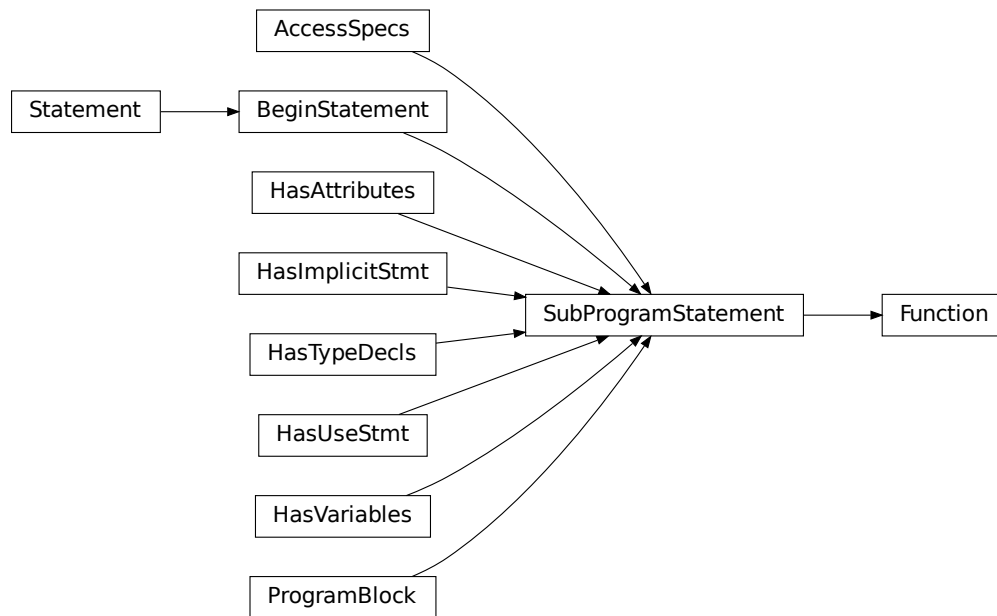
[<prefix>] FUNCTION <name> ( [<dummy-arg-list>] ) [<suffix>] <prefix> = <prefix-spec> [<prefix-spec>]...  
 <prefix-spec> = <declaration-type-spec>

RECURSIVE | PURE | ELEMENTAL

<suffix> = <proc-language-binding-spec> [RESULT ( <result-name> )]

RESULT ( <result-name> ) [<proc-language-binding-spec>]

## Inheritance



### `end_stmt_cls`

alias of *EndFunction*

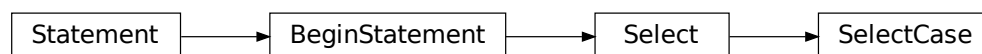
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** `fparser.one.block_statements.SelectCase`(*parent, item=None*)

[<case-construct-name> :] SELECT CASE ( <case-expr> )

## Inheritance



### `get_classes()`

Return the list of classes that this instance may have as children



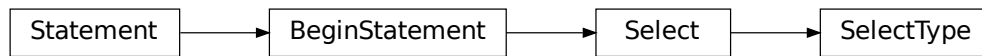
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**SelectType**(*parent, item=None*)

[<case-construct-name> :] SELECT TYPE ( <case-expr> )

### Inheritance



**get\_classes**()

Return the list of classes that this instance may have as children

**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

fparser.one.block\_statements.**WhereConstruct**

alias of **Where**

fparser.one.block\_statements.**ForallConstruct**

alias of **Forall**

**class** fparser.one.block\_statements.**IfThen**(*parent, item=None*)

[<if-construct-name> :] IF ( <scalar-logical-expr> ) THEN

**IfThen instance has the following attributes:**

*expr*

### Inheritance



**end\_stmt\_cls**

alias of *EndIfThen*

**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**process\_item()**

Process the line

**class** fparser.one.block\_statements.If(*parent*, *item=None*)

IF ( &lt;scalar-logical-expr&gt; ) action-stmt

**Inheritance****match**(*pos=0*, *endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**process\_item()**

Process the line

**class** fparser.one.block\_statements.Do(*parent*, *item=None*)

[&lt;do-construct-name&gt; :] DO label [loopcontrol] [&lt;do-construct-name&gt; :] DO [loopcontrol]

**Inheritance****end\_stmt\_cls**alias of [EndDo](#)**item\_re**(*pos=0*, *endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**match**(*pos=0*, *endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**process\_item()**

Parses the next line assuming it is a “Do” statement.

Overrides method in *BeginStatement*.

**process\_subitem**(*item*)

Check if item is blocks start statement, if it is, read the block.

Return True to stop adding items to given block.

**class** fparser.one.block\_statements.**Associate**(*parent*, *item=None*)

[<associate-construct-name> :] ASSOCIATE ( <association-list> )  
<block>

<association> = <associate-name> => <selector> <selector> = <expr> | <variable>

### Inheritance



**end\_stmt\_cls**

alias of *EndAssociate*

**match**(*pos=0*, *endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**process\_item**()

Process the line

fparser.one.block\_statements.**TypeDecl**

alias of Type

**class** fparser.one.block\_statements.**Enum**(*parent*, *item=None*)

**ENUM**, **BIND**(C)

<enumerator-def-stmt> [<enumerator-def-stmt>]...

### Inheritance



**end\_stmt\_cls**

alias of *EndEnum*

**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**process\_item**()

Process the line

**class** fparser.one.block\_statements.**EndSource**(*parent, item*)

Dummy End statement for BeginSource.

### Inheritance



**class** fparser.one.block\_statements.**EndModule**(*parent, item*)

### Inheritance

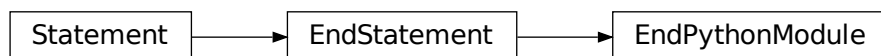


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**EndPythonModule**(*parent, item*)

### Inheritance

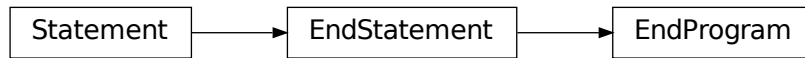


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.EndProgram(parent, item)
    END [PROGRAM [name]]
```

### Inheritance

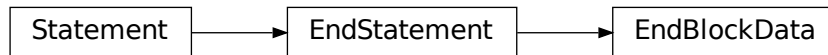


```
match(pos=0, endpos=9223372036854775807)
```

Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.EndBlockData(parent, item)
    END [BLOCK DATA [<block-data-name>]]
```

### Inheritance

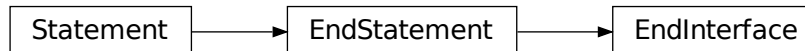


```
match(pos=0, endpos=9223372036854775807)
```

Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.EndInterface(parent, item)
```

### Inheritance

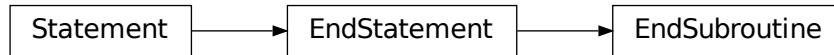


```
match(pos=0, endpos=9223372036854775807)
```

Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.EndSubroutine(parent, item)
    END [SUBROUTINE [name]]
```

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.EndFunction(parent, item)
END [FUNCTION [name]]
```

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.EndSelect(parent, item)
```

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.EndWhere(parent, item)
END WHERE [<where-construct-name>]
```

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**EndForall**(*parent, item*)  
 END FORALL [<forall-construct-name>]

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**EndIfThen**(*parent, item*)  
 END IF [<if-construct-name>]

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**EndDo**(*parent, item*)  
 END DO [<do-construct-name>]

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**process\_item**()

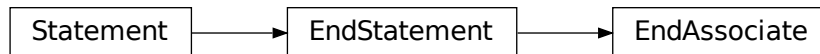
Parses the next line assuming it is an “End do” statement.

Overrides method in *EndStatement*.

**class** `fparser.one.block_statements.EndAssociate`(*parent, item*)

END ASSOCIATE [<associate-construct-name>]

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** `fparser.one.block_statements.EndType`(*parent, item*)

END TYPE [<type-name>]

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.



```
class fparser.one.block_statements.EndEnum(parent, item)
    END ENUM
```

### Inheritance

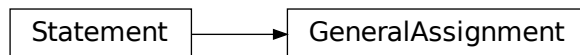


```
match(pos=0, endpos=9223372036854775807)
```

Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.GeneralAssignment(parent, item)
    <variable> = <expr> <pointer variable> => <expr>
```

### Inheritance



```
item_re(pos=0, endpos=9223372036854775807)
```

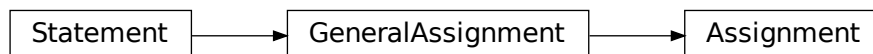
Matches zero or more characters at the beginning of the string.

```
match(pos=0, endpos=9223372036854775807)
```

Matches zero or more characters at the beginning of the string.

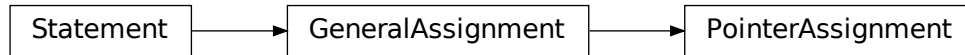
```
class fparser.one.block_statements.Assignment(parent, item)
```

### Inheritance



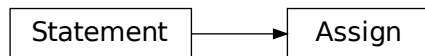
```
class fparser.one.block_statements.PointerAssignment(parent, item)
```

## Inheritance



```
class fparser.one.block_statements.Assign(parent, item)
    ASSIGN <label> TO <int-variable-name>
```

## Inheritance



```
match(pos=0, endpos=9223372036854775807)
```

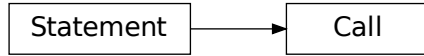
Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.Call(parent, item)
    Call statement class CALL <procedure-designator> [ ( [ <actual-arg-spec-list> ] ) ]
    <procedure-designator> = <procedure-name>

    <proc-component-ref>
    <data-ref> % <binding-name>
    <actual-arg-spec> = [ <keyword> = ] <actual-arg> <actual-arg> = <expr>
    <variable>
    <procedure-name>
    <proc-component-ref>
    <alt-return-spec>
    <alt-return-spec> = * <label>
    <proc-component-ref> = <variable> % <procedure-component-name>
    <variable> = <designator>

    Call instance has attributes:
        designator arg_list
```

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**process\_item**()

Parse the string containing the Call and store the designator and list of arguments (if any)

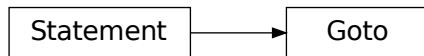
**tofortran**(*isfix=None*)

Returns the Fortran representation of this object as a string

**class** `fparser.one.block_statements.Goto`(*parent, item*)

GO TO <label>

## Inheritance



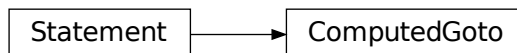
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** `fparser.one.block_statements.ComputedGoto`(*parent, item*)

GO TO ( <label-list> ) [ , ] <scalar-int-expr>

## Inheritance



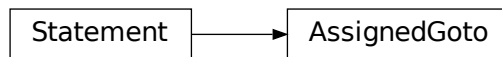
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**AssignedGoto**(*parent, item*)

GO TO <int-variable-name> [ ( <label> [ , <label> ]... ) ]

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Continue**(*parent, item*)

CONTINUE

### Inheritance



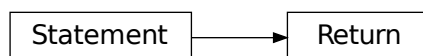
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Return**(*parent, item*)

RETURN [ <scalar-int-expr> ]

### Inheritance



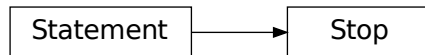
**match**(*pos*=0, *endpos*=9223372036854775807)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Stop**(*parent*, *item*)

STOP [ <stop-code> ] <stop-code> = <scalar-char-constant> | <1-5-digit>

### Inheritance



**match**(*pos*=0, *endpos*=9223372036854775807)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Print**(*parent*, *item*)

PRINT <format> [, <output-item-list>] <format> = <default-char-expr> | <label> | \*

<output-item> = <expr> | <io-implied-do> <io-implied-do> = ( <io-implied-do-object-list> , <implied-do-control> ) <io-implied-do-object> = <input-item> | <output-item> <implied-do-control> = <do-variable>

= <scalar-int-expr> ,  
 <scalar-int-expr> [ , <scalar-int-expr> ]

<input-item> = <variable> | <io-implied-do>

### Inheritance



**match**(*pos*=0, *endpos*=9223372036854775807)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Read**(*parent*, *item*)

Read0: READ ( <io-control-spec-list> ) [ <input-item-list> ]

<io-control-spec-list> = [ UNIT = ] <io-unit>

[ FORMAT = ] <format>

[ NML = ] <namelist-group-name>

ADVANCE = <scalar-default-char-expr>

...

**Read1: READ <format> [, <input-item-list>]**

<format> == <default-char-expr> | <label> | \*

### Inheritance

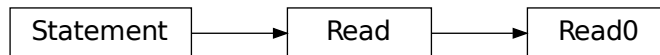


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Read0**(*parent, item*)

### Inheritance



**class** fparser.one.block\_statements.**Read1**(*parent, item*)

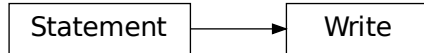
### Inheritance



**class** fparser.one.block\_statements.**Write**(*parent, item*)

WRITE ( io-control-spec-list ) [<output-item-list>]

## Inheritance



**match**(*pos*=0, *endpos*=9223372036854775807)

Matches zero or more characters at the beginning of the string.

**class** `fparser.one.block_statements.Flush`(*parent*, *item*)

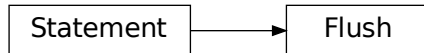
FLUSH <file-unit-number> FLUSH ( <flush-spec-list> ) <flush-spec> = [ UNIT = ] <file-unit-number>

IOSTAT = <scalar-int-variable>

IOMSG = <iomsg-variable>

ERR = <label>

## Inheritance



**match**(*pos*=0, *endpos*=9223372036854775807)

Matches zero or more characters at the beginning of the string.

**class** `fparser.one.block_statements.Wait`(*parent*, *item*)

WAIT ( <wait-spec-list> ) <wait-spec> = [ UNIT = ] <file-unit-number>

END = <label>

EOR = <label>

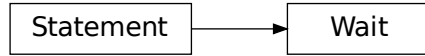
ERR = <label>

ID = <scalar-int-expr>

IOMSG = <iomsg-variable>

IOSTAT = <scalar-int-variable>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Contains**(*parent, item*)  
CONTAINS

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Allocate**(*parent, item*)  
ALLOCATE ( [ <type-spec> :: ] <allocation-list> [ , <alloc-opt-list> ] ) <alloc-opt> = STAT = <stat-variable>  
ERRMSG = <errmsg-variable>  
SOURCE = <source-expr>  
<allocation> = <allocate-object> [ ( <allocate-shape-spec-list> ) ]

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.



**process\_item()**

Process the ALLOCATE statement and store the various entities being allocated in self.items. Any type-specification is stored in self.spec.

**Raises**

**ParseError** – if an invalid type-specification is used

**tofortran(isfix=None)**

Create the Fortran code for this ALLOCATE statement

**Parameters**

**isfix** (*bool*) – whether or not to generate fixed-format code

**Returns**

Fortran code

**Return type**

str

**class** fparser.one.block\_statements.**Deallocate**(parent, item)

DEALLOCATE ( <allocate-object-list> [ , <dealloc-opt-list> ] ) <allocate-object> = <variable-name>

<structure-component>

<structure-component> = <data-ref> <dealloc-opt> = STAT = <stat-variable>

ERRMSG = <errmsg-variable>

**Inheritance**

**match**(pos=0, endpos=9223372036854775807)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**ModuleProcedure**(parent, item)

[ MODULE ] PROCEDURE [::] <procedure-name-list>

**Inheritance**

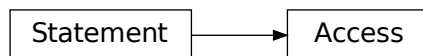
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Access**(*parent, item*)

<access-spec> [ [::] <access-id-list>] <access-spec> = PUBLIC | PRIVATE <access-id> = <use-name> |  
<generic-spec>

### Inheritance

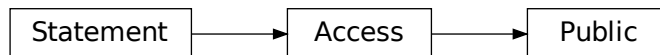


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

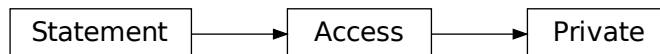
**class** fparser.one.block\_statements.**Public**(*parent, item*)

### Inheritance



**class** fparser.one.block\_statements.**Private**(*parent, item*)

### Inheritance



**class** fparser.one.block\_statements.**Close**(*parent, item*)

CLOSE ( <close-spec-list> ) <close-spec> = [ UNIT = ] <file-unit-number>

IOSTAT = <scalar-int-variable>

IOMSG = <iomsg-variable>  
 ERR = <label>  
 STATUS = <scalar-default-char-expr>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Cycle**(*parent, item*)

CYCLE [ <do-construct-name> ]

### Inheritance

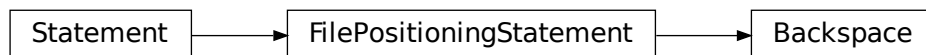


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

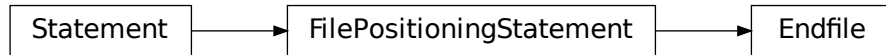
**class** fparser.one.block\_statements.**Backspace**(*parent, item*)

### Inheritance



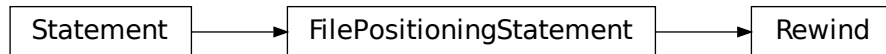
**class** fparser.one.block\_statements.**Endfile**(*parent, item*)

### Inheritance



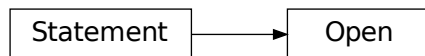
```
class fparser.one.block_statements.Rewind(parent, item)
```

### Inheritance



```
class fparser.one.block_statements.Open(parent, item)
    OPEN ( <connect-spec-list> ) <connect-spec> = [ UNIT = ] <file-unit-number>
    ACCESS = <scalar-default-char-expr>
    ..
```

### Inheritance



```
match(pos=0, endpos=9223372036854775807)
    Matches zero or more characters at the beginning of the string.
```

```
class fparser.one.block_statements.Format(parent, item)
    FORMAT <format-specification> <format-specification> = ( [ <format-item-list> ] ) <format-item> = [ <r> ]
    <data-edit-descr>
    <control-edit-descr>
    <char-string-edit-descr>
    [ <r> ] ( <format-item-list> )
```

**<data-edit-descr> = I <w> [ . <m> ]**

B <w> [ . <m> ]

...

<r|w|m|d|e> = <int-literal-constant> <v> = <signed-int-literal-constant> <control-edit-descr> = <position-edit-descr>

[ <r> ] /

:

...

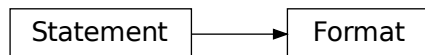
**<position-edit-descr> = T <n>**

TL <n>

...

<sign-edit-descr> = SS | SP | S ...

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Save**(*parent, item*)

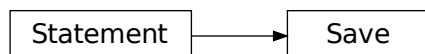
SAVE [ [ :: ] <saved-entity-list> ] <saved-entity> = <object-name>

<proc-pointer-name>

/ <common-block-name> /

<proc-pointer-name> = <name> <object-name> = <name>

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** `fparser.one.block_statements.Data`(*parent, item*)

DATA <data-stmt-set> [ [ , ] <data-stmt-set> ]... <data-stmt-set> = <data-stmt-object-list> / <data-stmt-value-list> / <data-stmt-object> = <variable> | <data-implied-do> <data-implied-do> = ( <data-i-do-object-list> , <data-i-do-variable> = <scalar-int-expr> , <scalar-int-expr> [ , <scalar-int-expr> ] )

<data-i-do-object> = <array-element>

<scalar-structure-component>

<data-implied-do>

<data-i-do-variable> = <scalar-int-variable> <variable> = <designator> <designator> = <object-name>

<array-element>

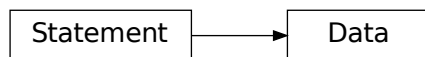
<array-section>

<structure-component>

<substring>

<array-element> = <data-ref> <array-section> = <data-ref> [ ( <substring-range> ) ]

## Inheritance



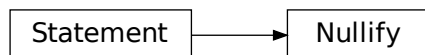
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** `fparser.one.block_statements.Nullify`(*parent, item*)

NULLIFY ( <pointer-object-list> ) <pointer-object> = <variable-name>

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Use**(*parent, item*)

Parses USE statement.

#### Parameters

**Statement** (*class*) – Base fparser class.

#### Raises

**AnalyzeError** – If entity name is not in module.

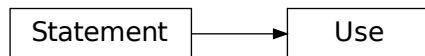
Fortran syntax construct:

USE [ [ , <module-nature> ] :: ] <module-name> [ , <rename-list> ] USE [ [ , <module-nature> ] :: ] <module-name> , ONLY : [ <only-list> ] <module-nature> = INTRINSIC | NON\_INTRINSIC <rename> = <local-name> => <use-name>

OPERATOR ( <local-defined-operator> ) => OPERATOR ( <use-defined-operator> )

<only> = <generic-spec> | <only-use-name> | <rename> <only-use-name> = <use-name>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**populate\_use\_provides**(*all\_mod\_provides, use\_provides, name, rename=None*)

Checks for entity name in the module

**process\_item**()

Parse the string containing the Use and store the module name and list of attributes (if any)

**tofortran**(*isfix=None*)

Returns the Fortran representation of this object as a string

#### Parameters

**isfix** (*bool*) – Whether or not to generated fixed-format Fortran

#### Returns

Fortran representation of this object

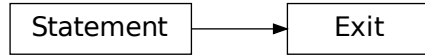
#### Return type

str

**class** fparser.one.block\_statements.**Exit**(*parent, item*)

EXIT [ <do-construct-name> ]

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Parameter**(*parent, item*)

PARAMETER ( <named-constant-def-list> ) <named-constant-def> = <named-constant> = <initialization-expr>

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Equivalence**(*parent, item*)

EQUIVALENCE <equivalence-set-list> <equivalence-set> = ( <equivalence-object> , <equivalence-object-list>  
) <equivalence-object> = <variable-name> | <array-element> | <substring>

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Dimension**(*parent, item*)

DIMENSION [ :: ] <array-name> ( <array-spec> )  
[ , <array-name> ( <array-spec> ) ]...



## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Target**(*parent, item*)

**TARGET** [ :: ] <object-name> ( <array-spec> )  
 [ , <object-name> ( <array-spec> ) ]...

## Inheritance



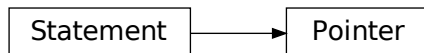
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Pointer**(*parent, item*)

**POINTER** [ :: ] <pointer-decl-list> <pointer-decl> = <object-name> [ ( <deferred-shape-spec-list> ) ]  
 <proc-entity-name>

## Inheritance



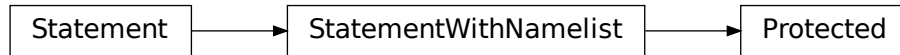
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Protected**(*parent, item*)

**PROTECTED** [ :: ] <entity-name-list>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Volatile**(*parent, item*)  
VOLATILE [ :: ] <object-name-list>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Value**(*parent, item*)  
VALUE [ :: ] <dummy-arg-name-list>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**ArithmeticIf**(*parent, item*)  
IF ( <scalar-numeric-expr> ) <label> , <label> , <label>

## Inheritance



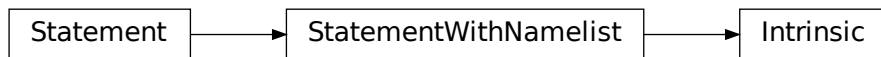
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Intrinsic**(*parent, item*)

INTRINSIC [ :: ] <intrinsic-procedure-name-list>

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Inquire**(*parent, item*)

INQUIRE ( <inquire-spec-list> ) INQUIRE ( IOLENGTH = <scalar-int-variable> ) <output-item-list>

<inquire-spec> = [ UNIT = ] <file-unit-number>

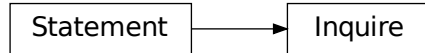
FILE = <file-name-expr>

...

<output-item> = <expr>

<io-implied-do>

## Inheritance

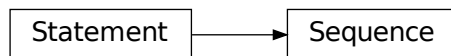


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.Sequence(parent, item)
SEQUENCE
```

## Inheritance

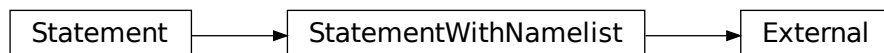


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.External(parent, item)
EXTERNAL [ :: ] <external-name-list>
```

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

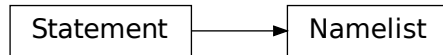
Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.Namelist(parent, item)
```

```
NAMELIST / <namelist-group-name> / <namelist-group-object-list> [ [ , ]
/ <namelist-group-name> / <namelist-group-object-list> ]...
```

```
<namelist-group-object> = <variable-name>
```

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Common**(*parent, item*)

```

COMMON [ / [ <common-block-name> ] / ] <common-block-object-list> [ [ , ] / [ <common-block-name> ] /
<common-block-object-list> ]... <common-block-object> = <variable-name> [ ( <explicit-shape-spec-list> ) ]
<proc-pointer-name>
  
```

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Optional**(*parent, item*)

```

OPTIONAL [ :: ] <dummy-arg-name-list> <dummy-arg-name> = <name>
  
```

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

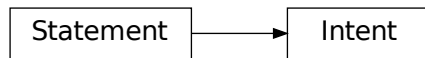
Matches zero or more characters at the beginning of the string.

**class** `fparser.one.block_statements.Intent`(*parent, item*)

INTENT ( <intent-spec> ) [ :: ] <dummy-arg-name-list> <intent-spec> = IN | OUT | INOUT

generalization for pyf-files: INTENT ( <intent-spec-list> ) [ :: ] <dummy-arg-name-list> <intent-spec> = IN | OUT | INOUT | CACHE | HIDE | OUT = <name>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** `fparser.one.block_statements.Entry`(*parent, item*)

ENTRY <entry-name> [ ( [ <dummy-arg-list> ] ) [ <suffix> ] ] <suffix> = <proc-language-binding-spec> [ RESULT ( <result-name> ) ]

RESULT ( <result-name> ) [ <proc-language-binding-spec> ]

<proc-language-binding-spec> = <language-binding-spec> <language-binding-spec> =

BIND ( C [ , NAME = <scalar-char-initialization-expr> ] )

<dummy-arg> = <dummy-arg-name> | \*

### Inheritance



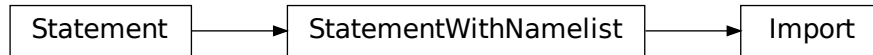
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** `fparser.one.block_statements.Import`(*parent, item*)

IMPORT [ [ :: ] <import-name-list> ]

## Inheritance



**match**(*pos*=0, *endpos*=9223372036854775807)

Matches zero or more characters at the beginning of the string.

`fparser.one.block_statements.ForallStmt`

alias of `Forall`

**class** `fparser.one.block_statements.SpecificBinding`(*parent*, *item*)

**PROCEDURE** [ ( <interface-name> ) ] [ [ , <binding-attr-list> ] :: ]

<binding-name> [ => <procedure-name> ]

<binding-attr> = **PASS** [ ( <arg-name> ) ]

**NOPASS**

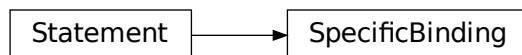
**NON\_OVERRIDABLE**

**DEFERRED**

<access-spec>

<access-spec> = **PUBLIC** | **PRIVATE**

## Inheritance



**match**(*pos*=0, *endpos*=9223372036854775807)

Matches zero or more characters at the beginning of the string.

**class** `fparser.one.block_statements.GenericBinding`(*parent*, *item*)

**GENERIC** [ , <access-spec> ] :: <generic-spec> => <binding-name-list>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**FinalBinding**(*parent, item*)

FINAL [ :: ] <final-subroutine-name-list>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

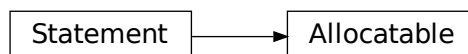
**class** fparser.one.block\_statements.**Allocatable**(*parent, item*)

ALLOCATABLE [ :: ] <object-name> [ ( <deferred-shape-spec-list> ) ]

[ , <object-name>  
[ ( <deferred-shape-spec-list> ) ]

]....

### Inheritance



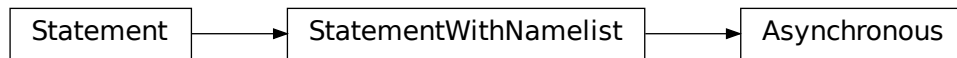
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.



```
class fparser.one.block_statements.Asynchronous(parent, item)
    ASYNCHRONOUS [ :: ] <object-name-list>
```

#### Inheritance



```
match(pos=0, endpos=9223372036854775807)
```

Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.Bind(parent, item)
    <language-binding-spec> [ :: ] <bind-entity-list> <language-binding-spec> =
        BIND ( C [ , NAME = <scalar-char-initialization-expr> ] )
    <bind-entity> = <entity-name> | / <common-block-name> /
```

#### Inheritance

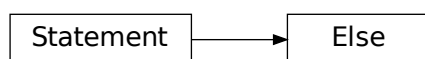


```
match(pos=0, endpos=9223372036854775807)
```

Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.Else(parent, item)
    ELSE [ <if-construct-name>]
```

#### Inheritance



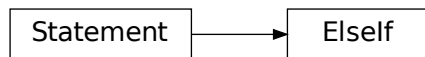
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.ElseIf(*parent, item*)

ELSE IF ( <scalar-logical-expr> ) THEN [ <if-construct-name> ]

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.Case(*parent, item*)

CASE <case-selector> [ <case-construct-name> ] <case-selector> = ( <case-value-range-list> ) | DEFAULT  
<case-value-range> = <case-value>

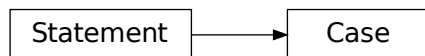
<case-value> :

: <case-value>

<case-value> : <case-value>

<case-value> = <scalar-(int|char|logical)-initialization-expr>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**process\_item**()

Populate the state of this item by parsing the associated line of code

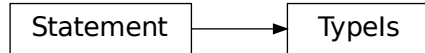
**tofortran**(*isfix=None*)

Return the Fortran for this object as a string

**class** fparser.one.block\_statements.TypeIs(*parent, item*)

TYPE IS <type-selector> [ <case-construct-name> ] <type-selector> = ( <type-value-range-list> ) <type-value-range> = <case-value> <case-value> = <char>

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**process\_item**()

Populate the state of this object by parsing the associated line of code

**tofortran**(*isfix=None*)

Create the Fortran representation of this object and return it as a string

**class** `fparser.one.block_statements.ClassIs`(*parent, item*)

CLASS <class-selector> <class-selector> = ( IS <type-value-range-list> | DEFAULT ) <type-value-range> =  
<case-value> <case-value> = <char>

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**process\_item**()

Populate the state of this object by parsing the string

**tofortran**(*isfix=None*)

Returns the Fortran for this object as a string

`fparser.one.block_statements.WhereStmt`

alias of `Where`

**class** `fparser.one.block_statements.ElseWhere`(*parent, item*)

ELSE WHERE ( <mask-expr> ) [ <where-construct-name> ] ELSE WHERE [ <where-construct-name> ]

## Inheritance



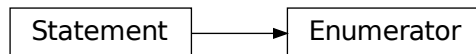
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Enumerator**(*parent, item*)

ENUMERATOR [ :: ] <enumerator-list> <enumerator> = <named-constant> [ = <scalar-int-initialization-expr> ]

## Inheritance



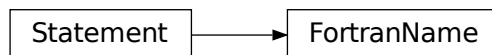
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**FortranName**(*parent, item*)

FORTRANNAME <name>

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Threadsafe**(*parent, item*)

THREADSAFE

## Inheritance

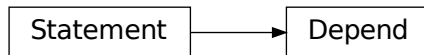


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.Depend(parent, item)
DEPEND ( <name-list> ) [ :: ] <dummy-arg-name-list>
```

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.Check(parent, item)
CHECK ( <c-int-scalar-expr> ) [ :: ] <name>
```

## Inheritance

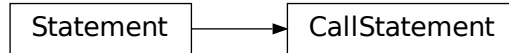


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.CallStatement(parent, item)
CALLSTATEMENT <c-expr>
```

## Inheritance

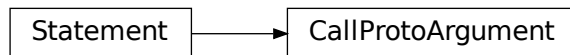


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**CallProtoArgument**(*parent, item*)  
CALLPROTOARGUMENT <c-type-spec-list>

## Inheritance

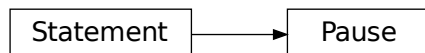


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Pause**(*parent, item*)  
PAUSE [ <char-literal-constant>[int-literal-constant> ]

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

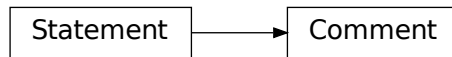
Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Comment**(*parent, item*)  
Attributes

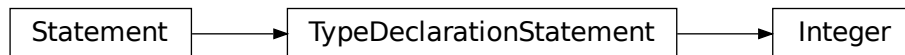
**content**  
[str] Content of the comment.

**is\_blank**

[bool] When True then Comment represents blank line.

**Inheritance**

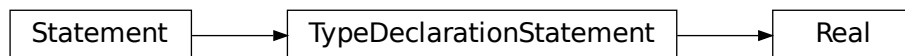
```
class fparser.one.block_statements.Integer(parent, item)
```

**Inheritance**

```
match(pos=0, endpos=9223372036854775807)
```

Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.Real(parent, item)
```

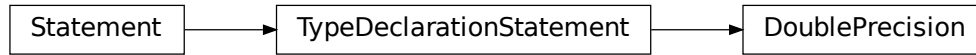
**Inheritance**

```
match(pos=0, endpos=9223372036854775807)
```

Matches zero or more characters at the beginning of the string.

```
class fparser.one.block_statements.DoublePrecision(parent, item)
```

## Inheritance

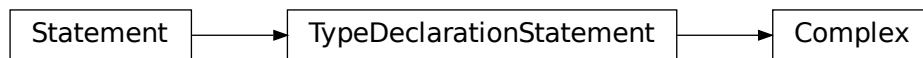


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Complex**(*parent, item*)

## Inheritance

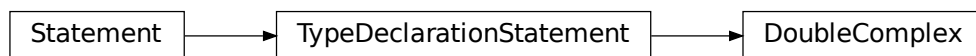


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**DoubleComplex**(*parent, item*)

## Inheritance



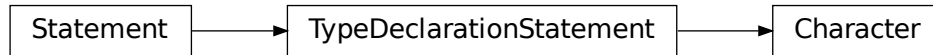
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Character**(*parent, item*)



## Inheritance

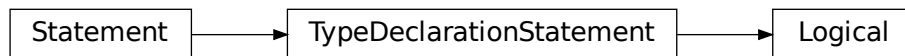


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** `fparser.one.block_statements.Logical`(*parent, item*)

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** `fparser.one.block_statements.Byte`(*parent, item*)

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

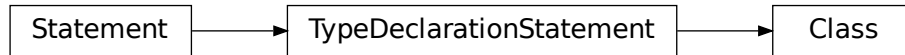
Matches zero or more characters at the beginning of the string.

`fparser.one.block_statements.TypeStmt`

alias of Type

**class** `fparser.one.block_statements.Class`(*parent, item*)

## Inheritance



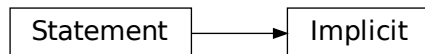
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.block\_statements.**Implicit**(*parent, item*)

IMPLICIT <implicit-spec-list> IMPLICIT NONE <implicit-spec> = <declaration-type-spec> ( <letter-spec-list> ) <letter-spec> = <letter> [ - <letter> ]

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

## Variables

- *intrinsic\_type\_spec*
- *declaration\_type\_spec*

fparser.one.block\_statements.**intrinsic\_type\_spec**

Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

```
[<class 'fparser.one.block_statements.SubprogramPrefix'>,
 <class 'fparser.one.typedecl_statements.Integer'>,
 <class 'fparser.one.typedecl_statements.Real'>,
 <class 'fparser.one.typedecl_statements.DoublePrecision'>,
 <class 'fparser.one.typedecl_statements.Complex'>,
 <class 'fparser.one.typedecl_statements.DoubleComplex'>,
 <class 'fparser.one.typedecl_statements.Character'>,
 <class 'fparser.one.typedecl_statements.Logical'>,
 <class 'fparser.one.typedecl_statements.Byte'>]
```

**fparser.one.block\_statements.declaration\_type\_spec**

Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

```
[<class 'fparser.one.block_statements.SubprogramPrefix'>,
 <class 'fparser.one.typedekl_statements.Integer'>,
 <class 'fparser.one.typedekl_statements.Real'>,
 <class 'fparser.one.typedekl_statements.DoublePrecision'>,
 <class 'fparser.one.typedekl_statements.Complex'>,
 <class 'fparser.one.typedekl_statements.DoubleComplex'>,
 <class 'fparser.one.typedekl_statements.Character'>,
 <class 'fparser.one.typedekl_statements.Logical'>,
 <class 'fparser.one.typedekl_statements.Byte'>,
 <class 'fparser.one.typedekl_statements.Type'>,
 <class 'fparser.one.typedekl_statements.Class'>]
```

**fparser.one.parsefortran**

Provides FortranParser.

- *Classes*

**Classes**

- *FortranParser*: Parser of FortranReader structure.

**class** fparser.one.parsefortran.**FortranParser**(reader, ignore\_comments=True)

Parser of FortranReader structure.

Use .parse() method for parsing, parsing result is saved in .block attribute.

**Inheritance**

FortranParser

**analyze()**

Attempts to analyse the parsed Fortran. It is not clear what for.

**get\_item()**

Retrieves the next item from the reader.

**parse()**

Parses the program specified in the reader object.

**put\_item(item)**

Pushes the given item to the reader.

**fparser.one.statements**

Fortran single line statements.

- *Classes*

**Classes**

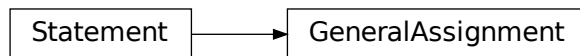
- *GeneralAssignment*: <variable> = <expr>
- *Assignment*: <variable> = <expr>
- *PointerAssignment*: <variable> = <expr>
- *Assign*: ASSIGN <label> TO <int-variable-name>
- *Call*: Call statement class
- *Goto*: GO TO <label>
- *ComputedGoto*: GO TO ( <label-list> ) [ , ] <scalar-int-expr>
- *AssignedGoto*: GO TO <int-variable-name> [ ( <label> [ , <label> ]... ) ]
- *Continue*: CONTINUE
- *Return*: RETURN [ <scalar-int-expr> ]
- *Stop*: STOP [ <stop-code> ]
- *Print*: PRINT <format> [ , <output-item-list> ]
- *Read*: Read0: READ ( <io-control-spec-list> ) [ <input-item-list> ]
- *Read0*: Read0: READ ( <io-control-spec-list> ) [ <input-item-list> ]
- *Read1*: Read0: READ ( <io-control-spec-list> ) [ <input-item-list> ]
- *Write*: WRITE ( io-control-spec-list ) [ <output-item-list> ]
- *Flush*: FLUSH <file-unit-number>
- *Wait*: WAIT ( <wait-spec-list> )
- *Contains*: CONTAINS
- *Allocate*: ALLOCATE ( [ <type-spec> :: ] <allocation-list> [ , <alloc-opt-list> ] )
- *Deallocate*: DEALLOCATE ( <allocate-object-list> [ , <dealloc-opt-list> ] )
- *ModuleProcedure*: [ MODULE ] PROCEDURE [::] <procedure-name-list>
- *Access*: <access-spec> [ [::] <access-id-list> ]
- *Public*: <access-spec> [ [::] <access-id-list> ]
- *Private*: <access-spec> [ [::] <access-id-list> ]

- *Close*: CLOSE ( <close-spec-list> )
- *Cycle*: CYCLE [ <do-construct-name> ]
- *Backspace*: REWIND <file-unit-number>
- *Endfile*: REWIND <file-unit-number>
- *Rewind*: REWIND <file-unit-number>
- *Open*: OPEN ( <connect-spec-list> )
- *Format*: FORMAT <format-specification>
- *Save*: SAVE [ [ :: ] <saved-entity-list> ]
- *Data*: DATA <data-stmt-set> [ [ , ] <data-stmt-set> ]...
- *Nullify*: NULLIFY ( <pointer-object-list> )
- *Use*: Parses USE statement.
- *Exit*: EXIT [ <do-construct-name> ]
- *Parameter*: PARAMETER ( <named-constant-def-list> )
- *Equivalence*: EQUIVALENCE <equivalence-set-list>
- *Dimension*: DIMENSION [ :: ] <array-name> ( <array-spec> )
- *Target*: TARGET [ :: ] <object-name> ( <array-spec> )
- *Pointer*: POINTER [ :: ] <pointer-decl-list>
- *Protected*: PROTECTED [ :: ] <entity-name-list>
- *Volatile*: VOLATILE [ :: ] <object-name-list>
- *Value*: VALUE [ :: ] <dummy-arg-name-list>
- *ArithmeticIf*: IF ( <scalar-numeric-expr> ) <label> , <label> , <label>
- *Intrinsic*: INTRINSIC [ :: ] <intrinsic-procedure-name-list>
- *Inquire*: INQUIRE ( <inquire-spec-list> )
- *Sequence*: SEQUENCE
- *External*: EXTERNAL [ :: ] <external-name-list>
- *Namelist*: NAMELIST / <namelist-group-name> / <namelist-group-object-list> [ [ , ]
- *Common*: COMMON [ / [ <common-block-name> ] / ] <common-block-object-list> [ [ , ] / [ <common-block-name> ] / <common-block-object-list> ]...
- *Optional*: OPTIONAL [ :: ] <dummy-arg-name-list>
- *Intent*: INTENT ( <intent-spec> ) [ :: ] <dummy-arg-name-list>
- *Entry*: ENTRY <entry-name> [ ( [ <dummy-arg-list> ] ) [ <suffix> ] ]
- *Import*: IMPORT [ [ :: ] <import-name-list> ]
- *ForallStmt*: FORALL <forall-header> <forall-assignment-stmt>
- *SpecificBinding*: PROCEDURE [ ( <interface-name> ) ] [ [ , <binding-attr-list> ] :: ]
- *GenericBinding*: GENERIC [ , <access-spec> ] :: <generic-spec> => <binding-name-list>
- *FinalBinding*: FINAL [ :: ] <final-subroutine-name-list>

- *Allocatable*: ALLOCATABLE [ :: ] <object-name> [ ( ( <deferred-shape-spec-list> ) ) ]
- *Asynchronous*: ASYNCHRONOUS [ :: ] <object-name-list>
- *Bind*: <language-binding-spec> [ :: ] <bind-entity-list>
- *Else*: ELSE [ <if-construct-name> ]
- *ElseIf*: ELSE IF ( <scalar-logical-expr> ) THEN [ <if-construct-name> ]
- *Case*: CASE <case-selector> [ <case-construct-name> ]
- *TypeIs*: TYPE IS <type-selector> [ <case-construct-name> ]
- *ClassIs*: CLASS <class-selector>
- *WhereStmt*: WHERE ( <mask-expr> ) <where-assignment-stmt>
- *ElseWhere*: ELSE WHERE ( <mask-expr> ) [ <where-construct-name> ]
- *Enumerator*: ENUMERATOR [ :: ] <enumerator-list>
- *FortranName*: FORTRANNAME <name>
- *Threadsafe*: THREADSAFE
- *Depend*: DEPEND ( <name-list> ) [ :: ] <dummy-arg-name-list>
- *Check*: CHECK ( <c-int-scalar-expr> ) [ :: ] <name>
- *CallStatement*: CALLSTATEMENT <c-expr>
- *CallProtoArgument*: CALLPROTOARGUMENT <c-type-spec-list>
- *Pause*: PAUSE [ <char-literal-constant> [ int-literal-constant> ]
- *Comment*: Attributes

```
class fparser.one.statements.GeneralAssignment(parent, item)
    <variable> = <expr> <pointer variable> => <expr>
```

### Inheritance



```
item_re(pos=0, endpos=9223372036854775807)
```

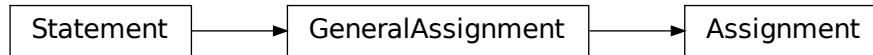
Matches zero or more characters at the beginning of the string.

```
match(pos=0, endpos=9223372036854775807)
```

Matches zero or more characters at the beginning of the string.

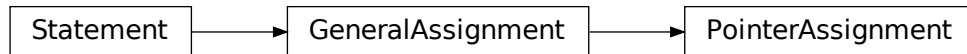
```
class fparser.one.statements.Assignment(parent, item)
```

## Inheritance



```
class fparser.one.statements.PointerAssignment(parent, item)
```

## Inheritance



```
class fparser.one.statements.Assign(parent, item)
  ASSIGN <label> TO <int-variable-name>
```

## Inheritance



```
match(pos=0, endpos=9223372036854775807)
```

Matches zero or more characters at the beginning of the string.

```
class fparser.one.statements.Call(parent, item)
```

```
Call statement class CALL <procedure-designator> [ ( [ <actual-arg-spec-list> ] ) ]
```

```
<procedure-designator> = <procedure-name>
```

```
<proc-component-ref>
```

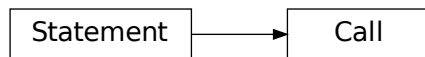
```
<data-ref> % <binding-name>
```

```
<actual-arg-spec> = [ <keyword> = ] <actual-arg> <actual-arg> = <expr>
```

```
<variable>
```

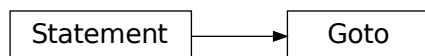
```
<procedure-name>
<proc-component-ref>
<alt-return-spec>
<alt-return-spec> = * <label>
<proc-component-ref> = <variable> % <procedure-component-name>
<variable> = <designator>
Call instance has attributes:
    designator arg_list
```

### Inheritance



```
match(pos=0, endpos=9223372036854775807)
    Matches zero or more characters at the beginning of the string.
process_item()
    Parse the string containing the Call and store the designator and list of arguments (if any)
tofortran(isfix=None)
    Returns the Fortran representation of this object as a string
class fparser.one.statements.Goto(parent, item)
    GO TO <label>
```

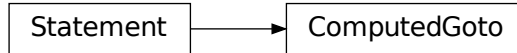
### Inheritance



```
match(pos=0, endpos=9223372036854775807)
    Matches zero or more characters at the beginning of the string.
class fparser.one.statements.ComputedGoto(parent, item)
    GO TO ( <label-list> ) [ , ] <scalar-int-expr>
```



## Inheritance

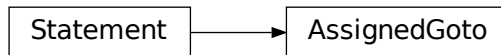


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**AssignedGoto**(*parent, item*)  
 GO TO <int-variable-name> [ ( <label> [ , <label> ]... ) ]

## Inheritance

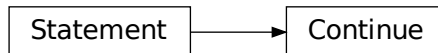


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Continue**(*parent, item*)  
 CONTINUE

## Inheritance

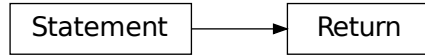


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Return**(*parent, item*)  
 RETURN [ <scalar-int-expr> ]

## Inheritance



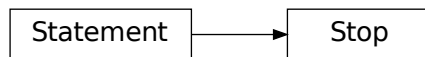
**match**(*pos*=0, *endpos*=9223372036854775807)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Stop**(*parent*, *item*)

STOP [ <stop-code> ] <stop-code> = <scalar-char-constant> | <1-5-digit>

## Inheritance



**match**(*pos*=0, *endpos*=9223372036854775807)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Print**(*parent*, *item*)

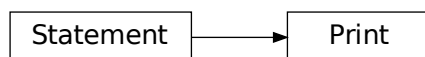
PRINT <format> [, <output-item-list>] <format> = <default-char-expr> | <label> | \*

<output-item> = <expr> | <io-implied-do> <io-implied-do> = ( <io-implied-do-object-list> , <implied-do-control> ) <io-implied-do-object> = <input-item> | <output-item> <implied-do-control> = <do-variable>

= <scalar-int-expr> ,  
    <scalar-int-expr> [ , <scalar-int-expr> ]

<input-item> = <variable> | <io-implied-do>

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Read**(*parent, item*)

Read0: READ ( <io-control-spec-list> ) [ <input-item-list> ]

<io-control-spec-list> = [ UNIT = ] <io-unit>

[ FORMAT = ] <format>

[ NML = ] <namelist-group-name>

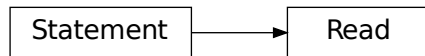
ADVANCE = <scalar-default-char-expr>

...

**Read1: READ <format> [, <input-item-list>]**

<format> == <default-char-expr> | <label> | \*

### Inheritance

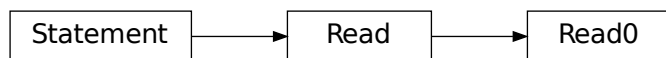


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Read0**(*parent, item*)

### Inheritance



**class** fparser.one.statements.**Read1**(*parent, item*)

### Inheritance



```
class fparser.one.statements.Write(parent, item)
    WRITE ( io-control-spec-list ) [<output-item-list>]
```

### Inheritance



```
match(pos=0, endpos=9223372036854775807)
    Matches zero or more characters at the beginning of the string.
```

```
class fparser.one.statements.Flush(parent, item)
    FLUSH <file-unit-number> FLUSH ( <flush-spec-list> ) <flush-spec> = [ UNIT = ] <file-unit-number>
    IOSTAT = <scalar-int-variable>
    IOMSG = <iomsg-variable>
    ERR = <label>
```

### Inheritance



```
match(pos=0, endpos=9223372036854775807)
    Matches zero or more characters at the beginning of the string.
```

```
class fparser.one.statements.Wait(parent, item)
    WAIT ( <wait-spec-list> ) <wait-spec> = [ UNIT = ] <file-unit-number>
    END = <label>
```

EOR = <label>  
 ERR = <label>  
 ID = <scalar-int-expr>  
 IOMSG = <iomsg-variable>  
 IOSTAT = <scalar-int-variable>

### Inheritance

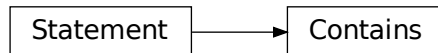


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Contains**(*parent, item*)  
 CONTAINS

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Allocate**(*parent, item*)  
 ALLOCATE ( [ <type-spec> :: ] <allocation-list> [ , <alloc-opt-list> ] ) <alloc-opt> = STAT = <stat-variable>  
 ERRMSG = <errmsg-variable>  
 SOURCE = <source-expr>  
 <allocation> = <allocate-object> [ ( <allocate-shape-spec-list> ) ]

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**process\_item**()

Process the ALLOCATE statement and store the various entities being allocated in self.items. Any type-specification is stored in self.spec.

### Raises

**ParseError** – if an invalid type-specification is used

**tofortran**(*isfix=None*)

Create the Fortran code for this ALLOCATE statement

### Parameters

**isfix** (*bool*) – whether or not to generate fixed-format code

### Returns

Fortran code

### Return type

str

**class** fparser.one.statements.**Deallocate**(*parent, item*)

DEALLOCATE ( <allocate-object-list> [ , <dealloc-opt-list> ] ) <allocate-object> = <variable-name>

<structure-component>

<structure-component> = <data-ref> <dealloc-opt> = STAT = <stat-variable>

ERRMSG = <errmsg-variable>

## Inheritance

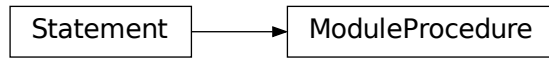


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

```
class fparser.one.statements.ModuleProcedure(parent, item)
    [ MODULE ] PROCEDURE [::] <procedure-name-list>
```

#### Inheritance



```
match(pos=0, endpos=9223372036854775807)
```

Matches zero or more characters at the beginning of the string.

```
class fparser.one.statements.Access(parent, item)
    <access-spec> [ [::] <access-id-list>] <access-spec> = PUBLIC | PRIVATE <access-id> = <use-name> |
    <generic-spec>
```

#### Inheritance

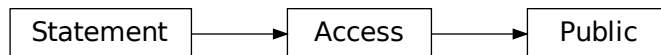


```
match(pos=0, endpos=9223372036854775807)
```

Matches zero or more characters at the beginning of the string.

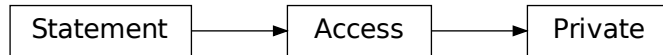
```
class fparser.one.statements.Public(parent, item)
```

#### Inheritance



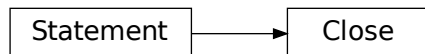
```
class fparser.one.statements.Private(parent, item)
```

### Inheritance



```
class fparser.one.statements.Close(parent, item)
    CLOSE ( <close-spec-list> ) <close-spec> = [ UNIT = ] <file-unit-number>
        IOSTAT = <scalar-int-variable>
        IOMSG = <iomsg-variable>
        ERR = <label>
        STATUS = <scalar-default-char-expr>
```

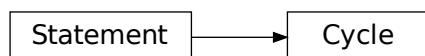
### Inheritance



```
match(pos=0, endpos=9223372036854775807)
    Matches zero or more characters at the beginning of the string.
```

```
class fparser.one.statements.Cycle(parent, item)
    CYCLE [ <do-construct-name> ]
```

### Inheritance

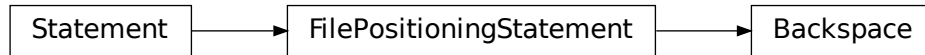


```
match(pos=0, endpos=9223372036854775807)
    Matches zero or more characters at the beginning of the string.
```

```
class fparser.one.statements.Backspace(parent, item)
```

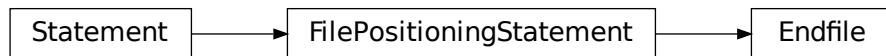


### Inheritance



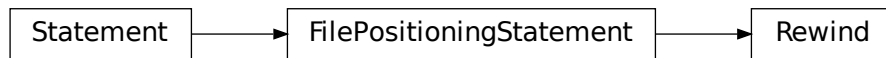
```
class fparser.one.statements.Endfile(parent, item)
```

### Inheritance



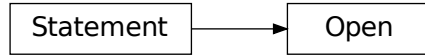
```
class fparser.one.statements.Rewind(parent, item)
```

### Inheritance



```
class fparser.one.statements.Open(parent, item)
  OPEN ( <connect-spec-list> ) <connect-spec> = [ UNIT = ] <file-unit-number>
    ACCESS = <scalar-default-char-expr>
    ..
```

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Format**(*parent, item*)

FORMAT <format-specification> <format-specification> = ( [ <format-item-list> ] ) <format-item> = [ <r> ]  
<data-edit-descr>

<control-edit-descr>

<char-string-edit-descr>

[ <r> ] ( <format-item-list> )

<data-edit-descr> = I <w> [ . <m> ]

B <w> [ . <m> ]

...

<r|w|m|d|e> = <int-literal-constant> <v> = <signed-int-literal-constant> <control-edit-descr> = <position-edit-descr>

[ <r> ] /

:

...

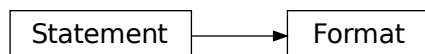
<position-edit-descr> = T <n>

TL <n>

...

<sign-edit-descr> = SS | SP | S ...

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Save**(*parent, item*)

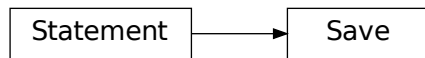
SAVE [ [ :: ] <saved-entity-list> ] <saved-entity> = <object-name>

<proc-pointer-name>

/ <common-block-name> /

<proc-pointer-name> = <name> <object-name> = <name>

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Data**(*parent, item*)

DATA <data-stmt-set> [ [ , ] <data-stmt-set> ]... <data-stmt-set> = <data-stmt-object-list> / <data-stmt-value-list> / <data-stmt-object> = <variable> | <data-implied-do> <data-implied-do> = ( <data-i-do-object-list> ,

<data-i-do-variable> = <scalar-int-expr> , <scalar-int-expr> [ , <scalar-int-expr> ] )

**<data-i-do-object> = <array-element>**

<scalar-structure-component>

<data-implied-do>

<data-i-do-variable> = <scalar-int-variable> <variable> = <designator> <designator> = <object-name>

<array-element>

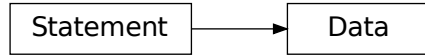
<array-section>

<structure-component>

<substring>

<array-element> = <data-ref> <array-section> = <data-ref> [ ( <substring-range> ) ]

## Inheritance



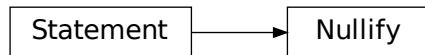
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Nullify**(*parent, item*)

NULLIFY ( <pointer-object-list> ) <pointer-object> = <variable-name>

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Use**(*parent, item*)

Parses USE statement.

### Parameters

**Statement** (*class*) – Base fparser class.

### Raises

**AnalyzeError** – If entity name is not in module.

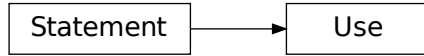
Fortran syntax construct:

USE [ [ , <module-nature> ] :: ] <module-name> [ , <rename-list> ] USE [ [ , <module-nature> ] :: ] <module-name> , ONLY : [ <only-list> ] <module-nature> = INTRINSIC | NON\_INTRINSIC <rename> = <local-name> => <use-name>

OPERATOR ( <local-defined-operator> ) => OPERATOR ( <use-defined-operator> )

<only> = <generic-spec> | <only-use-name> | <rename> <only-use-name> = <use-name>

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**populate\_use\_provides**(*all\_mod\_provides, use\_provides, name, rename=None*)

Checks for entity name in the module

**process\_item**()

Parse the string containing the Use and store the module name and list of attributes (if any)

**tofortran**(*isfix=None*)

Returns the Fortran representation of this object as a string

### Parameters

**isfix** (*bool*) – Whether or not to generated fixed-format Fortran

### Returns

Fortran representation of this object

### Return type

str

**class** fparser.one.statements.**Exit**(*parent, item*)

EXIT [ <do-construct-name> ]

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Parameter**(*parent, item*)

PARAMETER ( <named-constant-def-list> ) <named-constant-def> = <named-constant> = <initialization-expr>

## Inheritance



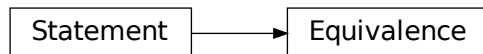
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Equivalence**(*parent, item*)

EQUIVALENCE <equivalence-set-list> <equivalence-set> = ( <equivalence-object> , <equivalence-object-list>  
) <equivalence-object> = <variable-name> | <array-element> | <substring>

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Dimension**(*parent, item*)

DIMENSION [ :: ] <array-name> ( <array-spec> )  
[ , <array-name> ( <array-spec> ) ]...

## Inheritance



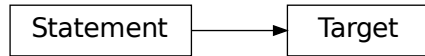
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

```
class fparser.one.statements.Target(parent, item)
```

```
TARGET [ :: ] <object-name> ( <array-spec> )  
    [ , <object-name> ( <array-spec> ) ]...
```

### Inheritance



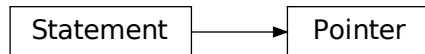
```
match(pos=0, endpos=9223372036854775807)
```

Matches zero or more characters at the beginning of the string.

```
class fparser.one.statements.Pointer(parent, item)
```

```
POINTER [ :: ] <pointer-decl-list> <pointer-decl> = <object-name> [ ( <deferred-shape-spec-list> ) ]  
    <proc-entity-name>
```

### Inheritance



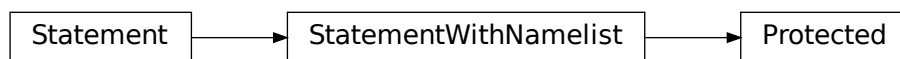
```
match(pos=0, endpos=9223372036854775807)
```

Matches zero or more characters at the beginning of the string.

```
class fparser.one.statements.Protected(parent, item)
```

```
PROTECTED [ :: ] <entity-name-list>
```

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Volatile**(*parent, item*)

VOLATILE [ :: ] <object-name-list>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Value**(*parent, item*)

VALUE [ :: ] <dummy-arg-name-list>

### Inheritance



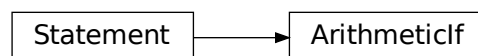
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**ArithmeticIf**(*parent, item*)

IF ( <scalar-numeric-expr> ) <label> , <label> , <label>

### Inheritance





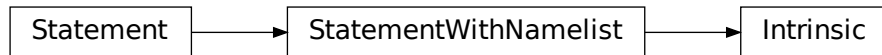
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Intrinsic**(*parent, item*)

INTRINSIC [ :: ] <intrinsic-procedure-name-list>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Inquire**(*parent, item*)

INQUIRE ( <inquire-spec-list> ) INQUIRE ( IOLENGTH = <scalar-int-variable> ) <output-item-list>

<inquire-spec> = [ UNIT = ] <file-unit-number>

FILE = <file-name-expr>

...

<output-item> = <expr>

<io-implied-do>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Sequence**(*parent, item*)

SEQUENCE

## Inheritance



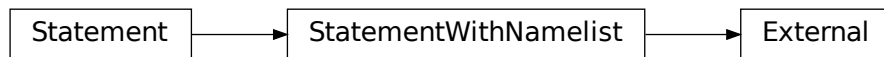
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**External**(*parent, item*)

EXTERNAL [ :: ] <external-name-list>

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

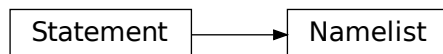
Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Namelist**(*parent, item*)

NAMELIST / <namelist-group-name> / <namelist-group-object-list> [ [ , ]  
/ <namelist-group-name> / <namelist-group-object-list> ]...

<namelist-group-object> = <variable-name>

## Inheritance



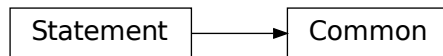
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

```
class fparser.one.statements.Common(parent, item)
```

```
COMMON [ / [ <common-block-name> ] / ] <common-block-object-list> [ [ , ] / [ <common-block-name> ] /  
<common-block-object-list> ]... <common-block-object> = <variable-name> [ ( <explicit-shape-spec-list> ) ]  
<proc-pointer-name>
```

### Inheritance



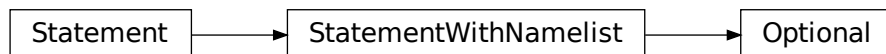
```
match(pos=0, endpos=9223372036854775807)
```

Matches zero or more characters at the beginning of the string.

```
class fparser.one.statements.Optional(parent, item)
```

```
OPTIONAL [ :: ] <dummy-arg-name-list> <dummy-arg-name> = <name>
```

### Inheritance



```
match(pos=0, endpos=9223372036854775807)
```

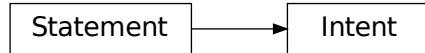
Matches zero or more characters at the beginning of the string.

```
class fparser.one.statements.Intent(parent, item)
```

```
INTENT ( <intent-spec> ) [ :: ] <dummy-arg-name-list> <intent-spec> = IN | OUT | INOUT
```

generalization for pyf-files: INTENT ( <intent-spec-list> ) [ :: ] <dummy-arg-name-list> <intent-spec> = IN | OUT | INOUT | CACHE | HIDE | OUT = <name>

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Entry**(*parent, item*)

ENTRY <entry-name> [ ( [ <dummy-arg-list> ] ) [ <suffix> ] ] <suffix> = <proc-language-binding-spec> [ RESULT ( <result-name> ) ]

RESULT ( <result-name> ) [ <proc-language-binding-spec> ]

<proc-language-binding-spec> = <language-binding-spec> <language-binding-spec> =

BIND ( C [ , NAME = <scalar-char-initialization-expr> ] )

<dummy-arg> = <dummy-arg-name> | \*

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Import**(*parent, item*)

IMPORT [ [ :: ] <import-name-list> ]

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**fparser.one.statements.ForallStmt**

alias of Forall

**class** fparser.one.statements.**SpecificBinding**(*parent, item*)

**PROCEDURE** [ ( <interface-name> ) ] [ [ , <binding-attr-list> ] :: ]

<binding-name> [ => <procedure-name> ]

<binding-attr> = **PASS** [ ( <arg-name> ) ]

**NOPASS**

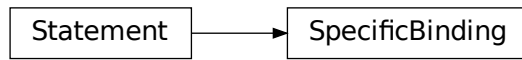
**NON\_OVERRIDABLE**

**DEFERRED**

<access-spec>

<access-spec> = **PUBLIC** | **PRIVATE**

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**GenericBinding**(*parent, item*)

**GENERIC** [ , <access-spec> ] :: <generic-spec> => <binding-name-list>

### Inheritance



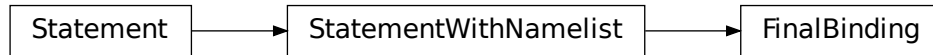
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**FinalBinding**(*parent, item*)

**FINAL** [ :: ] <final-subroutine-name-list>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Allocatable**(*parent, item*)

**ALLOCATABLE** [ :: ] <object-name> [ ( <deferred-shape-spec-list> ) ]

[ , <object-name>  
[ ( <deferred-shape-spec-list> ) ]

]....

### Inheritance



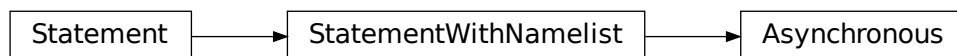
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Asynchronous**(*parent, item*)

**ASYNCHRONOUS** [ :: ] <object-name-list>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

```

class fparser.one.statements.Bind(parent, item)
    <language-binding-spec> [ :: ] <bind-entity-list> <language-binding-spec> =
        BIND ( C [ , NAME = <scalar-char-initialization-expr> ] )
    <bind-entity> = <entity-name> | / <common-block-name> /

```

### Inheritance



```

match(pos=0, endpos=9223372036854775807)

```

Matches zero or more characters at the beginning of the string.

```

class fparser.one.statements.Else(parent, item)
    ELSE [ <if-construct-name> ]

```

### Inheritance



```

match(pos=0, endpos=9223372036854775807)

```

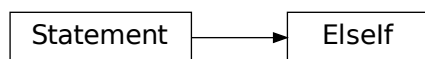
Matches zero or more characters at the beginning of the string.

```

class fparser.one.statements.ElseIf(parent, item)
    ELSE IF ( <scalar-logical-expr> ) THEN [ <if-construct-name> ]

```

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Case**(*parent, item*)

CASE <case-selector> [ <case-construct-name> ] <case-selector> = ( <case-value-range-list> ) | DEFAULT  
<case-value-range> = <case-value>

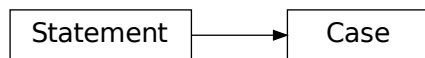
<case-value> :

: <case-value>

<case-value> : <case-value>

<case-value> = <scalar-(int|char|logical)-initialization-expr>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**process\_item**()

Populate the state of this item by parsing the associated line of code

**tofortran**(*isfix=None*)

Return the Fortran for this object as a string

**class** fparser.one.statements.**TypeIs**(*parent, item*)

TYPE IS <type-selector> [ <case-construct-name> ] <type-selector> = ( <type-value-range-list> ) <type-value-range> = <case-value> <case-value> = <char>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**process\_item**()

Populate the state of this object by parsing the associated line of code



**tofortran**(*isfix=None*)

Create the Fortran representation of this object and return it as a string

**class** fparser.one.statements.**ClassIs**(*parent, item*)

CLASS <class-selector> <class-selector> = ( IS <type-value-range-list> | DEFAULT ) <type-value-range> =  
<case-value> <case-value> = <char>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**process\_item**()

Populate the state of this object by parsing the string

**tofortran**(*isfix=None*)

Returns the Fortran for this object as a string

fparser.one.statements.**WhereStmt**

alias of Where

**class** fparser.one.statements.**ElseWhere**(*parent, item*)

ELSE WHERE ( <mask-expr> ) [ <where-construct-name> ] ELSE WHERE [ <where-construct-name> ]

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Enumerator**(*parent, item*)

ENUMERATOR [ :: ] <enumerator-list> <enumerator> = <named-constant> [ = <scalar-int-initialization-expr> ]

### Inheritance



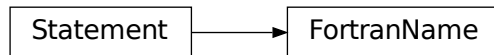
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**FortranName**(*parent, item*)

FORTRANNAME <name>

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Threadsafe**(*parent, item*)

THREADSAFE

### Inheritance



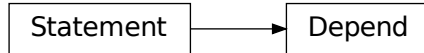
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Depend**(*parent, item*)

DEPEND ( <name-list> ) [ :: ] <dummy-arg-name-list>

## Inheritance



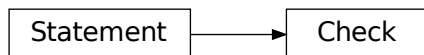
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Check**(*parent, item*)

CHECK ( <c-int-scalar-expr> ) [ :: ] <name>

## Inheritance



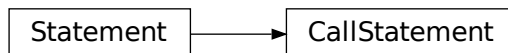
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**CallStatement**(*parent, item*)

CALLSTATEMENT <c-expr>

## Inheritance



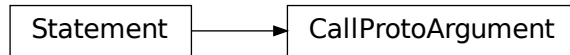
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**CallProtoArgument**(*parent, item*)

CALLPROTOARGUMENT <c-type-spec-list>

## Inheritance



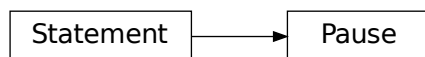
**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Pause**(*parent, item*)

PAUSE [ <char-literal-constant|int-literal-constant> ]

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.statements.**Comment**(*parent, item*)

Attributes

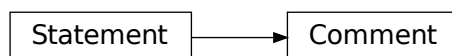
**content**

[str] Content of the comment.

**is\_blank**

[bool] When True then Comment represents blank line.

## Inheritance



**fparser.one.tests**

- *Submodules*

**Submodules****fparser.one.tests.test\_scripts**

Test fparser one scripts

**fparser.one.typedekl\_statements**

Fortran type-declaration statements.

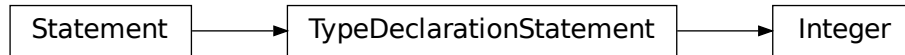
- *Classes*
- *Variables*

**Classes**

- *Integer*: <declaration-type-spec> [ [, <attr-spec> ] :: ] <entity-decl-list>
- *Real*: <declaration-type-spec> [ [, <attr-spec> ] :: ] <entity-decl-list>
- *DoublePrecision*: <declaration-type-spec> [ [, <attr-spec> ] :: ] <entity-decl-list>
- *Complex*: <declaration-type-spec> [ [, <attr-spec> ] :: ] <entity-decl-list>
- *DoubleComplex*: <declaration-type-spec> [ [, <attr-spec> ] :: ] <entity-decl-list>
- *Character*: <declaration-type-spec> [ [, <attr-spec> ] :: ] <entity-decl-list>
- *Logical*: <declaration-type-spec> [ [, <attr-spec> ] :: ] <entity-decl-list>
- *Byte*: <declaration-type-spec> [ [, <attr-spec> ] :: ] <entity-decl-list>
- *TypeStmt*: <declaration-type-spec> [ [, <attr-spec> ] :: ] <entity-decl-list>
- *Class*: <declaration-type-spec> [ [, <attr-spec> ] :: ] <entity-decl-list>
- *Implicit*: IMPLICIT <implicit-spec-list>

**class** fparser.one.typedekl\_statements.**Integer**(parent, item)

### Inheritance

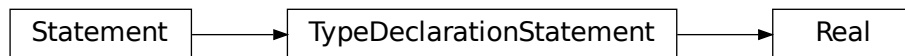


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** `fparser.one.typedekl_statements.Real`(*parent, item*)

### Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** `fparser.one.typedekl_statements.DoublePrecision`(*parent, item*)

### Inheritance

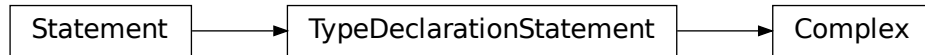


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** `fparser.one.typedekl_statements.Complex`(*parent, item*)

## Inheritance

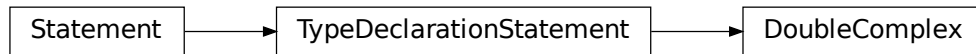


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.typedecl\_statements.**DoubleComplex**(*parent, item*)

## Inheritance

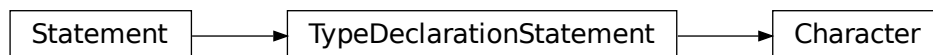


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.typedecl\_statements.**Character**(*parent, item*)

## Inheritance

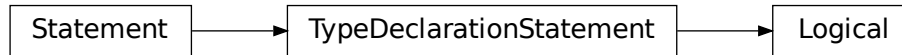


**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.typedecl\_statements.**Logical**(*parent, item*)

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

**class** fparser.one.typedekl\_statements.**Byte**(*parent, item*)

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

Matches zero or more characters at the beginning of the string.

fparser.one.typedekl\_statements.**TypeStmt**

alias of Type

**class** fparser.one.typedekl\_statements.**Class**(*parent, item*)

## Inheritance



**match**(*pos=0, endpos=9223372036854775807*)

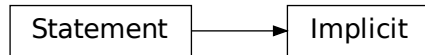
Matches zero or more characters at the beginning of the string.

**class** fparser.one.typedekl\_statements.**Implicit**(*parent, item*)

IMPLICIT <implicit-spec-list> IMPLICIT NONE <implicit-spec> = <declaration-type-spec> ( <letter-spec-list>  
) <letter-spec> = <letter> [ - <letter> ]



## Inheritance



**match**(pos=0, endpos=9223372036854775807)

Matches zero or more characters at the beginning of the string.

## Variables

- *intrinsic\_type\_spec*
- *declaration\_type\_spec*

`fparser.one.typedecl_statements.intrinsic_type_spec`

Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

```
[<class 'fparser.one.typedecl_statements.Integer'>,
 <class 'fparser.one.typedecl_statements.Real'>,
 <class 'fparser.one.typedecl_statements.DoublePrecision'>,
 <class 'fparser.one.typedecl_statements.Complex'>,
 <class 'fparser.one.typedecl_statements.DoubleComplex'>,
 <class 'fparser.one.typedecl_statements.Character'>,
 <class 'fparser.one.typedecl_statements.Logical'>,
 <class 'fparser.one.typedecl_statements.Byte'>]
```

`fparser.one.typedecl_statements.declaration_type_spec`

Built-in mutable sequence.

If no argument is given, the constructor creates a new empty list. The argument must be an iterable if specified.

```
[<class 'fparser.one.typedecl_statements.Integer'>,
 <class 'fparser.one.typedecl_statements.Real'>,
 <class 'fparser.one.typedecl_statements.DoublePrecision'>,
 <class 'fparser.one.typedecl_statements.Complex'>,
 <class 'fparser.one.typedecl_statements.DoubleComplex'>,
 <class 'fparser.one.typedecl_statements.Character'>,
 <class 'fparser.one.typedecl_statements.Logical'>,
 <class 'fparser.one.typedecl_statements.Byte'>,
 <class 'fparser.one.typedecl_statements.Type'>,
 <class 'fparser.one.typedecl_statements.Class'>]
```

### `fparser.scripts`

- *Submodules*

#### Submodules

##### `fparser.scripts.fparser2`

Example script to parse a Fortran program using fparser

##### `fparser.scripts.fparser2_bench`

Generates a large Fortran program in memory and then measures how long it takes fparser2 to parse it. This is based on the benchmark suggested by Ondřej Čertík via Ioannis Nikiteas.

##### `fparser.scripts.parse`

##### `fparser.scripts.read`

Python script with command line options which calls the Fortran File Reader with the supplied filename(s) and outputs the reader's representation of the code(s).

##### `fparser.scripts.script_options`

- *Functions*

#### Functions

- `set_read_options()`: Undocumented.
- `set_parse_options()`: Undocumented.
- `get_fortran_code_group()`: Undocumented.

`fparser.scripts.script_options.set_read_options(parser)`

`fparser.scripts.script_options.set_parse_options(parser)`

`fparser.scripts.script_options.get_fortran_code_group(parser)`

**fparser.two**

- *Submodules*

**Submodules****fparser.two.C99Preprocessor**

C99 Preprocessor Syntax Rules.

**fparser.two.Fortran2003**

Fortran 2003 Syntax Rules.

**fparser.two.Fortran2008**

The file implements the Fortran2008 rules as defined in <https://j3-fortran.org/doc/year/10/10-007r1.pdf>

**fparser.two.parser**

This file provides utilities to create a Fortran parser suitable for a particular standard.

**fparser.two.pattern\_tools**

Tools for constructing patterns.

Permission to use, modify, and distribute this software is given under the terms of the NumPy License. See <http://scipy.org>.

NO WARRANTY IS EXPRESSED OR IMPLIED. USE AT YOUR OWN RISK. Author: Pearu Peterson  
<pearu@cens.ioc.ee> Created: Oct 2006

**fparser.two.symbol\_table**

The fparser2 symbol-table module. Defines various classes as well as the single, global SYMBOL\_TABLES instance. The latter is a container for all of the top-level scoping units encountered during parsing.

- *Classes*
- *Exceptions*
- *Variables*

## Classes

- *SymbolTables*: Class encapsulating functionality for the global symbol-tables object.
- *SymbolTable*: Class implementing a single symbol table.

### `class fparser.two.symbol_table.SymbolTables`

Class encapsulating functionality for the global symbol-tables object. This is a container for all symbol tables constructed while parsing code. All names are converted to lower case (since Fortran is not case sensitive).

## Inheritance

SymbolTables

### `add(name)`

Add a new symbol table with the supplied name. The name will be converted to lower case if necessary.

#### Parameters

**name** (*str*) – the name for the new table.

#### Returns

the new symbol table.

#### Return type

*fparser.two.symbol\_table.SymbolTable*

#### Raises

*SymbolTableError* – if there is already an entry with the supplied name.

### `clear()`

Deletes any stored SymbolTables but retains the stored list of classes that define scoping units.

### property `current_scope`

#### Returns

the symbol table for the current scoping unit or None.

#### Return type

*fparser.two.symbol\_table.SymbolTable* or *NoneType*

### `enable_checks(value)`

Sets whether or not to enable consistency checks in every symbol table that is created during a parse.

#### Parameters

**value** (*bool*) – whether or not checks are enabled.

### `enter_scope(name)`

Called when the parser enters a new scoping region (i.e. when it encounters one of the classes listed in *\_scoping\_unit\_classes*). Sets the ‘current scope’ to be the symbol table with the supplied name. If we are not currently within a tree of scoping regions then a new entry is created in the internal dict of symbol tables. If there is an existing tree then a new table is created and added to the bottom.

**Parameters**

**name** (*str*) – name of the scoping region.

**exit\_scope()**

Marks the end of the processing of the current scoping unit. Since we are exiting the current scoping region, the new ‘current scoping region’ will be its parent.

**Raises**

*SymbolTableError* – if there is no current scope from which to exit.

**lookup(name)**

Find the named symbol table and return it.

**Parameters**

**name** (*str*) – the name of the required symbol table (not case sensitive).

**Returns**

the named symbol table.

**Return type**

*fparser.two.symbol\_table.SymbolTable*

**remove(name)**

Removes the named symbol table and any descendants it may have. When searching for the named table, the current scope takes priority followed by the list of top-level symbol tables.

**Parameters**

**name** (*str*) – the name of the symbol table to remove (not case sensitive).

**Raises**

*SymbolTableError* – if the named symbol table is not in the current scope or in the list of top-level symbol tables.

**property scoping\_unit\_classes****Returns**

the fparser2 classes that are taken to mark the start of a new scoping region.

**Return type**

list of types

**class** *fparser.two.symbol\_table.SymbolTable*(*name*, *parent=None*, *checking\_enabled=False*)

Class implementing a single symbol table.

Since this functionality is not yet fully mature, checks that new symbols don’t clash with existing symbols are disabled by default. Once #201 is complete it is planned to switch this so that the checks are instead enabled by default.

**Parameters**

- **name** (*str*) – the name of this scope. Will be the name of the associated module or routine.
- **parent** (*fparser.two.symbol\_table.SymbolTable.Symbol*) – the symbol table within which this one is nested (if any).
- **checking\_enabled** (*bool*) – whether or not validity checks are performed for symbols added to the table.

## Inheritance

SymbolTable

**class** `Symbol`(*name*, *primitive\_type*)

**property** `name`

Alias for field number 0

**property** `primitive_type`

Alias for field number 1

**add\_child**(*child*)

Adds a child symbol table (scoping region nested within this one).

**Parameters**

**child** (*fparser.two.symbol\_table.SymbolTable*) – the nested symbol table.

**Raises**

**TypeError** – if the supplied child is not a SymbolTable.

**add\_data\_symbol**(*name*, *primitive\_type*)

Creates a new Symbol with the specified properties and adds it to the symbol table. The supplied name is converted to lower case.

TODO #201 add support for other symbol properties (kind, shape and visibility).

**Parameters**

- **name** (*str*) – the name of the symbol.
- **primitive\_type** (*str*) – the primitive type of the symbol.

**Raises**

- **TypeError** – if any of the supplied parameters are of the wrong type.
- **SymbolTableError** – if the symbol table already contains an entry with the supplied name.

**add\_use\_symbols**(*name*, *only\_list*=None, *rename\_list*=None)

Creates an entry in the table for the USE of a module with the supplied name. If no *only\_list* is supplied then this USE represents a wildcard import of all public symbols in the named module. If the USE statement has an ONLY clause but without any named symbols then *only\_list* should be an empty list.

A USE can also have one or more rename entries *without* an only list.

**Parameters**

- **name** (*str*) – the name of the module being imported via a USE. Not case sensitive.
- **only\_list** (*Optional[List[Tuple[str, str | NoneType]]*) – if there is an ‘only:’ clause on the USE statement then this contains a list of tuples, each holding the local name of the symbol and its name in the module from which it is imported. These names are case insensitive.

- **rename\_list** (*Optional*[*List*[*Tuple*[*str*, *str*]]]) – a list of symbols that are re-named from the scope being imported. Each entry is a tuple containing the name in the local scope and the corresponding name in the module from which it is imported. These names are case insensitive.

### property children

#### Returns

the child (nested) symbol tables, if any.

#### Return type

list of *fparser.two.symbol\_table.SymbolTable*

### del\_child(*name*)

Removes the named symbol table.

#### Parameters

**name** (*str*) – the name of the child symbol table to delete (not case sensitive).

#### Raises

**KeyError** – if the named table is not a child of this one.

### lookup(*name*)

Lookup the symbol with the supplied name.

#### Parameters

**name** (*str*) – the name of the symbol to lookup (not case sensitive).

#### Returns

the named symbol.

#### Return type

*fparser.two.symbol\_table.SymbolTable.Symbol*

#### Raises

**KeyError** – if the named symbol cannot be found in this or any parent scope.

### property name

#### Returns

the name of this symbol table (scoping region).

#### Return type

*str*

### property parent

#### Returns

the parent symbol table (scoping region) that contains this one (if any).

#### Return type

*fparser.two.symbol\_table.SymbolTable* or *NoneType*

### property root

#### Returns

the top-level symbol table that contains the current scoping region (symbol table).

#### Return type

*fparser.two.symbol\_table.SymbolTable*

## Exceptions

- *SymbolTableError*: Base class exception for symbol-table related errors.

**exception** `fparser.two.symbol_table.SymbolTableError`

Base class exception for symbol-table related errors.

## Inheritance

SymbolTableError

## Variables

- *SYMBOL\_TABLES*

`fparser.two.symbol_table.SYMBOL_TABLES`

The single, global container for all symbol tables constructed while parsing.

<fparser.two.symbol\_table.SymbolTables object at 0x7ff8c170b8d0>

## `fparser.two.utils`

Base classes and exception handling for Fortran parser.

The Sphinx/AutoAPI documentation is [here](#). or you can use the navigation pane on the left.



## DOXYGEN-GENERATED DOCUMENTATION



## PYTHON MODULE INDEX

### f

- [fparser](#), 3
- [fparser.api](#), 3
- [fparser.common](#), 3
  - [fparser.common.base\\_classes](#), 3
  - [fparser.common.readfortran](#), 7
  - [fparser.common.sourceinfo](#), 13
  - [fparser.common.splitline](#), 13
  - [fparser.common.tests](#), 14
  - [fparser.common.tests.logging\\_utils](#), 14
  - [fparser.common.utils](#), 14
- [fparser.one](#), 17
  - [fparser.one.block\\_statements](#), 17
  - [fparser.one.parsefortran](#), 71
  - [fparser.one.statements](#), 72
  - [fparser.one.tests](#), 105
  - [fparser.one.tests.test\\_scripts](#), 105
  - [fparser.one.typedecl\\_statements](#), 105
- [fparser.scripts](#), 110
  - [fparser.scripts.fparser2](#), 110
  - [fparser.scripts.fparser2\\_bench](#), 110
  - [fparser.scripts.parse](#), 110
  - [fparser.scripts.read](#), 110
  - [fparser.scripts.script\\_options](#), 110
- [fparser.two](#), 111
  - [fparser.two.C99Preprocessor](#), 111
  - [fparser.two.Fortran2003](#), 111
  - [fparser.two.Fortran2008](#), 111
  - [fparser.two.parser](#), 111
  - [fparser.two.pattern\\_tools](#), 111
  - [fparser.two.symbol\\_table](#), 111
  - [fparser.two.utils](#), 116



## A

Access (class in *fparser.one.block\_statements*), 46  
 Access (class in *fparser.one.statements*), 83  
 add() (*fparser.two.symbol\_table.SymbolTables* method), 112  
 add\_child() (*fparser.two.symbol\_table.SymbolTable* method), 114  
 add\_data\_symbol() (*fparser.two.symbol\_table.SymbolTable* method), 114  
 add\_use\_symbols() (*fparser.two.symbol\_table.SymbolTable* method), 114  
 Allocatable (class in *fparser.one.block\_statements*), 60  
 Allocatable (class in *fparser.one.statements*), 98  
 Allocate (class in *fparser.one.block\_statements*), 44  
 Allocate (class in *fparser.one.statements*), 81  
 analyze() (*fparser.one.parsefortran.FortranParser* method), 71  
 AnalyzeError, 16  
 apply\_map() (*fparser.common.readfortran.Line* method), 10  
 ArithmeticIf (class in *fparser.one.block\_statements*), 54  
 ArithmeticIf (class in *fparser.one.statements*), 92  
 Assign (class in *fparser.one.block\_statements*), 38  
 Assign (class in *fparser.one.statements*), 75  
 AssignedGoto (class in *fparser.one.block\_statements*), 40  
 AssignedGoto (class in *fparser.one.statements*), 77  
 Assignment (class in *fparser.one.block\_statements*), 37  
 Assignment (class in *fparser.one.statements*), 74  
 Associate (class in *fparser.one.block\_statements*), 31  
 Asynchronous (class in *fparser.one.block\_statements*), 60  
 Asynchronous (class in *fparser.one.statements*), 98  
 AttributeHolder (class in *fparser.common.base\_classes*), 6

## B

Backspace (class in *fparser.one.block\_statements*), 47  
 Backspace (class in *fparser.one.statements*), 84  
 BeginSource (class in *fparser.one.block\_statements*), 20

BeginStatement (class in *fparser.common.base\_classes*), 4  
 Bind (class in *fparser.one.block\_statements*), 61  
 Bind (class in *fparser.one.statements*), 98  
 BlockData (class in *fparser.one.block\_statements*), 24  
 Byte (class in *fparser.one.block\_statements*), 69  
 Byte (class in *fparser.one.typeddecl\_statements*), 108

## C

Call (class in *fparser.one.block\_statements*), 38  
 Call (class in *fparser.one.statements*), 75  
 CallProtoArgument (class in *fparser.one.block\_statements*), 66  
 CallProtoArgument (class in *fparser.one.statements*), 103  
 CallStatement (class in *fparser.one.block\_statements*), 65  
 CallStatement (class in *fparser.one.statements*), 103  
 Case (class in *fparser.one.block\_statements*), 62  
 Case (class in *fparser.one.statements*), 100  
 Character (class in *fparser.one.block\_statements*), 68  
 Character (class in *fparser.one.typeddecl\_statements*), 107  
 Check (class in *fparser.one.block\_statements*), 65  
 Check (class in *fparser.one.statements*), 103  
 children (*fparser.two.symbol\_table.SymbolTable* property), 115  
 Class (class in *fparser.one.block\_statements*), 69  
 Class (class in *fparser.one.typeddecl\_statements*), 108  
 classes (class in *fparser.common.utils*), 15  
 ClassIs (class in *fparser.one.block\_statements*), 63  
 ClassIs (class in *fparser.one.statements*), 101  
 clear() (*fparser.two.symbol\_table.SymbolTables* method), 112  
 clone() (*fparser.common.readfortran.Line* method), 10  
 Close (class in *fparser.one.block\_statements*), 46  
 Close (class in *fparser.one.statements*), 84  
 close\_source() (*fparser.common.readfortran.FortranFileReader* method), 8  
 Comment (class in *fparser.common.readfortran*), 10  
 Comment (class in *fparser.one.block\_statements*), 66  
 Comment (class in *fparser.one.statements*), 104

Common (class in *fparser.one.block\_statements*), 57  
Common (class in *fparser.one.statements*), 94  
Complex (class in *fparser.one.block\_statements*), 68  
Complex (class in *fparser.one.typedcl\_statements*), 106  
ComputedGoto (class in *fparser.one.block\_statements*), 39  
ComputedGoto (class in *fparser.one.statements*), 76  
Contains (class in *fparser.one.block\_statements*), 44  
Contains (class in *fparser.one.statements*), 81  
Continue (class in *fparser.one.block\_statements*), 40  
Continue (class in *fparser.one.statements*), 77  
copy() (*fparser.common.readfortran.Line* method), 10  
current\_scope (*fparser.two.symbol\_table.SymbolTables* property), 112  
Cycle (class in *fparser.one.block\_statements*), 47  
Cycle (class in *fparser.one.statements*), 84

## D

Data (class in *fparser.one.block\_statements*), 50  
Data (class in *fparser.one.statements*), 87  
Deallocate (class in *fparser.one.block\_statements*), 45  
Deallocate (class in *fparser.one.statements*), 82  
declaration\_type\_spec (in module *fparser.one.block\_statements*), 70  
declaration\_type\_spec (in module *fparser.one.typedcl\_statements*), 109  
del\_child() (*fparser.two.symbol\_table.SymbolTable* method), 115  
Depend (class in *fparser.one.block\_statements*), 65  
Depend (class in *fparser.one.statements*), 102  
Dimension (class in *fparser.one.block\_statements*), 52  
Dimension (class in *fparser.one.statements*), 90  
Do (class in *fparser.one.block\_statements*), 30  
DoubleComplex (class in *fparser.one.block\_statements*), 68  
DoubleComplex (class in *fparser.one.typedcl\_statements*), 107  
DoublePrecision (class in *fparser.one.block\_statements*), 67  
DoublePrecision (class in *fparser.one.typedcl\_statements*), 106

## E

Else (class in *fparser.one.block\_statements*), 61  
Else (class in *fparser.one.statements*), 99  
ElseIf (class in *fparser.one.block\_statements*), 62  
ElseIf (class in *fparser.one.statements*), 99  
ElseWhere (class in *fparser.one.block\_statements*), 63  
ElseWhere (class in *fparser.one.statements*), 101  
enable\_checks() (*fparser.two.symbol\_table.SymbolTables* method), 112  
end\_stmt\_cls (*fparser.one.block\_statements.Associate* attribute), 31

end\_stmt\_cls (*fparser.one.block\_statements.BeginSource* attribute), 21  
end\_stmt\_cls (*fparser.one.block\_statements.BlockData* attribute), 25  
end\_stmt\_cls (*fparser.one.block\_statements.Do* attribute), 30  
end\_stmt\_cls (*fparser.one.block\_statements.Enum* attribute), 31  
end\_stmt\_cls (*fparser.one.block\_statements.Function* attribute), 28  
end\_stmt\_cls (*fparser.one.block\_statements.IfThen* attribute), 29  
end\_stmt\_cls (*fparser.one.block\_statements.Interface* attribute), 26  
end\_stmt\_cls (*fparser.one.block\_statements.Module* attribute), 22  
end\_stmt\_cls (*fparser.one.block\_statements.Program* attribute), 24  
end\_stmt\_cls (*fparser.one.block\_statements.PythonModule* attribute), 23  
end\_stmt\_cls (*fparser.one.block\_statements.Subroutine* attribute), 27  
EndAssociate (class in *fparser.one.block\_statements*), 36  
EndBlockData (class in *fparser.one.block\_statements*), 33  
EndDo (class in *fparser.one.block\_statements*), 35  
EndEnum (class in *fparser.one.block\_statements*), 36  
Endfile (class in *fparser.one.block\_statements*), 47  
Endfile (class in *fparser.one.statements*), 85  
EndForall (class in *fparser.one.block\_statements*), 35  
EndFunction (class in *fparser.one.block\_statements*), 34  
EndIfThen (class in *fparser.one.block\_statements*), 35  
EndInterface (class in *fparser.one.block\_statements*), 33  
EndModule (class in *fparser.one.block\_statements*), 32  
EndProgram (class in *fparser.one.block\_statements*), 32  
EndPythonModule (class in *fparser.one.block\_statements*), 32  
EndSelect (class in *fparser.one.block\_statements*), 34  
EndSource (class in *fparser.one.block\_statements*), 32  
EndStatement (class in *fparser.common.base\_classes*), 5  
EndSubroutine (class in *fparser.one.block\_statements*), 33  
EndType (class in *fparser.one.block\_statements*), 36  
EndWhere (class in *fparser.one.block\_statements*), 34  
enter\_scope() (*fparser.two.symbol\_table.SymbolTables* method), 112  
Entry (class in *fparser.one.block\_statements*), 58  
Entry (class in *fparser.one.statements*), 96  
Enum (class in *fparser.one.block\_statements*), 31  
Enumerator (class in *fparser.one.block\_statements*), 64  
Enumerator (class in *fparser.one.statements*), 101

- Equivalence (class in *fparser.one.block\_statements*), 52  
 Equivalence (class in *fparser.one.statements*), 90  
 Exit (class in *fparser.one.block\_statements*), 51  
 Exit (class in *fparser.one.statements*), 89  
 exit\_scope() (*fparser.two.symbol\_table.SymbolTables* method), 113  
 External (class in *fparser.one.block\_statements*), 56  
 External (class in *fparser.one.statements*), 94
- ## F
- fill() (*fparser.common.base\_classes.BeginStatement* method), 5  
 FinalBinding (class in *fparser.one.block\_statements*), 60  
 FinalBinding (class in *fparser.one.statements*), 97  
 Flush (class in *fparser.one.block\_statements*), 43  
 Flush (class in *fparser.one.statements*), 80  
 ForallConstruct (in module *fparser.one.block\_statements*), 29  
 ForallStmt (in module *fparser.one.block\_statements*), 59  
 ForallStmt (in module *fparser.one.statements*), 97  
 Format (class in *fparser.one.block\_statements*), 48  
 Format (class in *fparser.one.statements*), 86  
 FortranFileReader (class in *fparser.common.readfortran*), 8  
 FortranName (class in *fparser.one.block\_statements*), 64  
 FortranName (class in *fparser.one.statements*), 102  
 FortranParser (class in *fparser.one.parsefortran*), 71  
 FortranReaderError, 11  
 FortranStringReader (class in *fparser.common.readfortran*), 9  
 fparser  
   module, 3  
 fparser.api  
   module, 3  
 fparser.common  
   module, 3  
 fparser.common.base\_classes  
   module, 3  
 fparser.common.readfortran  
   module, 7  
 fparser.common.sourceinfo  
   module, 13  
 fparser.common.splitline  
   module, 13  
 fparser.common.tests  
   module, 14  
 fparser.common.tests.logging\_utils  
   module, 14  
 fparser.common.utils  
   module, 14  
 fparser.one  
   module, 17  
 fparser.one.block\_statements  
   module, 17  
 fparser.one.parsefortran  
   module, 71  
 fparser.one.statements  
   module, 72  
 fparser.one.tests  
   module, 105  
 fparser.one.tests.test\_scripts  
   module, 105  
 fparser.one.typeddecl\_statements  
   module, 105  
 fparser.scripts  
   module, 110  
 fparser.scripts.fparser2  
   module, 110  
 fparser.scripts.fparser2\_bench  
   module, 110  
 fparser.scripts.parse  
   module, 110  
 fparser.scripts.read  
   module, 110  
 fparser.scripts.script\_options  
   module, 110  
 fparser.two  
   module, 111  
 fparser.two.C99Preprocessor  
   module, 111  
 fparser.two.Fortran2003  
   module, 111  
 fparser.two.Fortran2008  
   module, 111  
 fparser.two.parser  
   module, 111  
 fparser.two.pattern\_tools  
   module, 111  
 fparser.two.symbol\_table  
   module, 111  
 fparser.two.utils  
   module, 116  
 Function (class in *fparser.one.block\_statements*), 27
- ## G
- GeneralAssignment (class in *fparser.one.block\_statements*), 37  
 GeneralAssignment (class in *fparser.one.statements*), 74  
 GenericBinding (class in *fparser.one.block\_statements*), 59  
 GenericBinding (class in *fparser.one.statements*), 97  
 get\_classes() (*fparser.one.block\_statements.SelectCase* method), 28  
 get\_classes() (*fparser.one.block\_statements.SelectType* method), 29

`get_fortran_code_group()` (in module `fparser.scripts.script_options`), 110  
`get_item()` (`fparser.one.parsefortran.FortranParser` method), 71  
`get_module_file()` (in module `fparser.common.utils`), 15  
`get_provides()` (`fparser.common.base_classes.Statement` method), 4  
`get_provides()` (`fparser.one.block_statements.Module` method), 22  
`get_type()` (`fparser.common.base_classes.Statement` method), 4  
`get_variable()` (`fparser.common.base_classes.Statement` method), 4  
`Goto` (class in `fparser.one.block_statements`), 39  
`Goto` (class in `fparser.one.statements`), 76

## H

`handle_unknown_item_and_raise()` (`fparser.common.base_classes.BeginStatement` method), 5  
`has_map()` (`fparser.common.readfortran.Line` method), 10

## I

`If` (class in `fparser.one.block_statements`), 30  
`IfThen` (class in `fparser.one.block_statements`), 29  
`Implicit` (class in `fparser.one.block_statements`), 70  
`Implicit` (class in `fparser.one.typeddecl_statements`), 108  
`Import` (class in `fparser.one.block_statements`), 58  
`Import` (class in `fparser.one.statements`), 96  
`Inquire` (class in `fparser.one.block_statements`), 55  
`Inquire` (class in `fparser.one.statements`), 93  
`Integer` (class in `fparser.one.block_statements`), 67  
`Integer` (class in `fparser.one.typeddecl_statements`), 105  
`Intent` (class in `fparser.one.block_statements`), 57  
`Intent` (class in `fparser.one.statements`), 95  
`Interface` (class in `fparser.one.block_statements`), 25  
`Intrinsic` (class in `fparser.one.block_statements`), 55  
`Intrinsic` (class in `fparser.one.statements`), 93  
`intrinsic_type_spec` (in module `fparser.one.block_statements`), 70  
`intrinsic_type_spec` (in module `fparser.one.typeddecl_statements`), 109  
`is_name` (in module `fparser.common.utils`), 16  
`isempty()` (`fparser.common.readfortran.Comment` method), 10  
`isempty()` (`fparser.common.readfortran.MultiLine` method), 11  
`item_re()` (`fparser.one.block_statements.Do` method), 30  
`item_re()` (`fparser.one.block_statements.GeneralAssignment` method), 37

`item_re()` (`fparser.one.statements.GeneralAssignment` method), 74

## L

`Line` (class in `fparser.common.readfortran`), 9  
`Logical` (class in `fparser.one.block_statements`), 69  
`Logical` (class in `fparser.one.typeddecl_statements`), 107  
`lookup()` (`fparser.two.symbol_table.SymbolTable` method), 115  
`lookup()` (`fparser.two.symbol_table.SymbolTables` method), 113

## M

`match()` (`fparser.one.block_statements.Access` method), 46  
`match()` (`fparser.one.block_statements.Allocatable` method), 60  
`match()` (`fparser.one.block_statements.Allocate` method), 44  
`match()` (`fparser.one.block_statements.ArithmeticIf` method), 55  
`match()` (`fparser.one.block_statements.Assign` method), 38  
`match()` (`fparser.one.block_statements.AssignedGoto` method), 40  
`match()` (`fparser.one.block_statements.Associate` method), 31  
`match()` (`fparser.one.block_statements.Asynchronous` method), 61  
`match()` (`fparser.one.block_statements.Bind` method), 61  
`match()` (`fparser.one.block_statements.BlockData` method), 25  
`match()` (`fparser.one.block_statements.Byte` method), 69  
`match()` (`fparser.one.block_statements.Call` method), 39  
`match()` (`fparser.one.block_statements.CallProtoArgument` method), 66  
`match()` (`fparser.one.block_statements.CallStatement` method), 66  
`match()` (`fparser.one.block_statements.Case` method), 62  
`match()` (`fparser.one.block_statements.Character` method), 69  
`match()` (`fparser.one.block_statements.Check` method), 65  
`match()` (`fparser.one.block_statements.Class` method), 70  
`match()` (`fparser.one.block_statements.ClassIs` method), 63  
`match()` (`fparser.one.block_statements.Close` method), 47  
`match()` (`fparser.one.block_statements.Common` method), 57  
`match()` (`fparser.one.block_statements.Complex` method), 68



`match()` (*fparser.one.block\_statements.ComputedGoto* method), 39  
`match()` (*fparser.one.block\_statements.Contains* method), 44  
`match()` (*fparser.one.block\_statements.Continue* method), 40  
`match()` (*fparser.one.block\_statements.Cycle* method), 47  
`match()` (*fparser.one.block\_statements.Data* method), 50  
`match()` (*fparser.one.block\_statements.Deallocate* method), 45  
`match()` (*fparser.one.block\_statements.Depend* method), 65  
`match()` (*fparser.one.block\_statements.Dimension* method), 53  
`match()` (*fparser.one.block\_statements.Do* method), 30  
`match()` (*fparser.one.block\_statements.DoubleComplex* method), 68  
`match()` (*fparser.one.block\_statements.DoublePrecision* method), 68  
`match()` (*fparser.one.block\_statements.Else* method), 61  
`match()` (*fparser.one.block\_statements.ElseIf* method), 62  
`match()` (*fparser.one.block\_statements.ElseWhere* method), 64  
`match()` (*fparser.one.block\_statements.EndAssociate* method), 36  
`match()` (*fparser.one.block\_statements.EndBlockData* method), 33  
`match()` (*fparser.one.block\_statements.EndDo* method), 36  
`match()` (*fparser.one.block\_statements.EndEnum* method), 37  
`match()` (*fparser.one.block\_statements.EndForall* method), 35  
`match()` (*fparser.one.block\_statements.EndFunction* method), 34  
`match()` (*fparser.one.block\_statements.EndIfThen* method), 35  
`match()` (*fparser.one.block\_statements.EndInterface* method), 33  
`match()` (*fparser.one.block\_statements.EndModule* method), 32  
`match()` (*fparser.one.block\_statements.EndProgram* method), 33  
`match()` (*fparser.one.block\_statements.EndPythonModule* method), 32  
`match()` (*fparser.one.block\_statements.EndSelect* method), 34  
`match()` (*fparser.one.block\_statements.EndSubroutine* method), 34  
`match()` (*fparser.one.block\_statements.EndType* method), 36  
`match()` (*fparser.one.block\_statements.EndWhere* method), 35  
`match()` (*fparser.one.block\_statements.Entry* method), 58  
`match()` (*fparser.one.block\_statements.Enum* method), 31  
`match()` (*fparser.one.block\_statements.Enumerator* method), 64  
`match()` (*fparser.one.block\_statements.Equivalence* method), 52  
`match()` (*fparser.one.block\_statements.Exit* method), 52  
`match()` (*fparser.one.block\_statements.External* method), 56  
`match()` (*fparser.one.block\_statements.FinalBinding* method), 60  
`match()` (*fparser.one.block\_statements.Flush* method), 43  
`match()` (*fparser.one.block\_statements.Format* method), 49  
`match()` (*fparser.one.block\_statements.FortranName* method), 64  
`match()` (*fparser.one.block\_statements.Function* method), 28  
`match()` (*fparser.one.block\_statements.GeneralAssignment* method), 37  
`match()` (*fparser.one.block\_statements.GenericBinding* method), 60  
`match()` (*fparser.one.block\_statements.Goto* method), 39  
`match()` (*fparser.one.block\_statements.If* method), 30  
`match()` (*fparser.one.block\_statements.IfThen* method), 29  
`match()` (*fparser.one.block\_statements.Implicit* method), 70  
`match()` (*fparser.one.block\_statements.Import* method), 59  
`match()` (*fparser.one.block\_statements.Inquire* method), 56  
`match()` (*fparser.one.block\_statements.Integer* method), 67  
`match()` (*fparser.one.block\_statements.Intent* method), 58  
`match()` (*fparser.one.block\_statements.Interface* method), 26  
`match()` (*fparser.one.block\_statements.Intrinsic* method), 55  
`match()` (*fparser.one.block\_statements.Logical* method), 69  
`match()` (*fparser.one.block\_statements.Module* method), 22  
`match()` (*fparser.one.block\_statements.ModuleProcedure* method), 45  
`match()` (*fparser.one.block\_statements.Namelist* method), 57  
`match()` (*fparser.one.block\_statements.Nullify* method), 50

`match()` (*fparser.one.block\_statements.Open method*), 48  
`match()` (*fparser.one.block\_statements.Optional method*), 57  
`match()` (*fparser.one.block\_statements.Parameter method*), 52  
`match()` (*fparser.one.block\_statements.Pause method*), 66  
`match()` (*fparser.one.block\_statements.Pointer method*), 53  
`match()` (*fparser.one.block\_statements.Print method*), 41  
`match()` (*fparser.one.block\_statements.Program method*), 24  
`match()` (*fparser.one.block\_statements.Protected method*), 54  
`match()` (*fparser.one.block\_statements.PythonModule method*), 23  
`match()` (*fparser.one.block\_statements.Read method*), 42  
`match()` (*fparser.one.block\_statements.Real method*), 67  
`match()` (*fparser.one.block\_statements.Return method*), 40  
`match()` (*fparser.one.block\_statements.Save method*), 49  
`match()` (*fparser.one.block\_statements.SelectCase method*), 28  
`match()` (*fparser.one.block\_statements.SelectType method*), 29  
`match()` (*fparser.one.block\_statements.Sequence method*), 56  
`match()` (*fparser.one.block\_statements.SpecificBinding method*), 59  
`match()` (*fparser.one.block\_statements.Stop method*), 41  
`match()` (*fparser.one.block\_statements.Subroutine method*), 27  
`match()` (*fparser.one.block\_statements.Target method*), 53  
`match()` (*fparser.one.block\_statements.ThreadSafe method*), 65  
`match()` (*fparser.one.block\_statements.TypeIs method*), 63  
`match()` (*fparser.one.block\_statements.Use method*), 51  
`match()` (*fparser.one.block\_statements.Value method*), 54  
`match()` (*fparser.one.block\_statements.Volatile method*), 54  
`match()` (*fparser.one.block\_statements.Wait method*), 44  
`match()` (*fparser.one.block\_statements.Write method*), 43  
`match()` (*fparser.one.statements.Access method*), 83  
`match()` (*fparser.one.statements.Allocatable method*), 98  
`match()` (*fparser.one.statements.Allocate method*), 82  
`match()` (*fparser.one.statements.ArithmeticIf method*), 92  
`match()` (*fparser.one.statements.Assign method*), 75  
`match()` (*fparser.one.statements.AssignedGoto method*), 77  
`match()` (*fparser.one.statements.Asynchronous method*), 98  
`match()` (*fparser.one.statements.Bind method*), 99  
`match()` (*fparser.one.statements.Call method*), 76  
`match()` (*fparser.one.statements.CallProtoArgument method*), 104  
`match()` (*fparser.one.statements.CallStatement method*), 103  
`match()` (*fparser.one.statements.Case method*), 100  
`match()` (*fparser.one.statements.Check method*), 103  
`match()` (*fparser.one.statements.ClassIs method*), 101  
`match()` (*fparser.one.statements.Close method*), 84  
`match()` (*fparser.one.statements.Common method*), 95  
`match()` (*fparser.one.statements.ComputedGoto method*), 77  
`match()` (*fparser.one.statements.Contains method*), 81  
`match()` (*fparser.one.statements.Continue method*), 77  
`match()` (*fparser.one.statements.Cycle method*), 84  
`match()` (*fparser.one.statements.Data method*), 88  
`match()` (*fparser.one.statements.Deallocate method*), 82  
`match()` (*fparser.one.statements.Depend method*), 103  
`match()` (*fparser.one.statements.Dimension method*), 90  
`match()` (*fparser.one.statements.Else method*), 99  
`match()` (*fparser.one.statements.ElseIf method*), 99  
`match()` (*fparser.one.statements.ElseWhere method*), 101  
`match()` (*fparser.one.statements.Entry method*), 96  
`match()` (*fparser.one.statements.Enumerator method*), 102  
`match()` (*fparser.one.statements.Equivalence method*), 90  
`match()` (*fparser.one.statements.Exit method*), 89  
`match()` (*fparser.one.statements.External method*), 94  
`match()` (*fparser.one.statements.FinalBinding method*), 98  
`match()` (*fparser.one.statements.Flush method*), 80  
`match()` (*fparser.one.statements.Format method*), 86  
`match()` (*fparser.one.statements.FortranName method*), 102  
`match()` (*fparser.one.statements.GeneralAssignment method*), 74  
`match()` (*fparser.one.statements.GenericBinding method*), 97  
`match()` (*fparser.one.statements.Goto method*), 76  
`match()` (*fparser.one.statements.Import method*), 96  
`match()` (*fparser.one.statements.Inquire method*), 93  
`match()` (*fparser.one.statements.Intent method*), 96  
`match()` (*fparser.one.statements.Intrinsic method*), 93  
`match()` (*fparser.one.statements.ModuleProcedure method*), 83  
`match()` (*fparser.one.statements.Namelist method*), 94  
`match()` (*fparser.one.statements.Nullify method*), 88  
`match()` (*fparser.one.statements.Open method*), 86

- `match()` (*fparser.one.statements.Optional method*), 95
  - `match()` (*fparser.one.statements.Parameter method*), 90
  - `match()` (*fparser.one.statements.Pause method*), 104
  - `match()` (*fparser.one.statements.Pointer method*), 91
  - `match()` (*fparser.one.statements.Print method*), 78
  - `match()` (*fparser.one.statements.Protected method*), 91
  - `match()` (*fparser.one.statements.Read method*), 79
  - `match()` (*fparser.one.statements.Return method*), 78
  - `match()` (*fparser.one.statements.Save method*), 87
  - `match()` (*fparser.one.statements.Sequence method*), 94
  - `match()` (*fparser.one.statements.SpecificBinding method*), 97
  - `match()` (*fparser.one.statements.Stop method*), 78
  - `match()` (*fparser.one.statements.Target method*), 91
  - `match()` (*fparser.one.statements.ThreadSafe method*), 102
  - `match()` (*fparser.one.statements.TypeIs method*), 100
  - `match()` (*fparser.one.statements.Use method*), 89
  - `match()` (*fparser.one.statements.Value method*), 92
  - `match()` (*fparser.one.statements.Volatile method*), 92
  - `match()` (*fparser.one.statements.Wait method*), 81
  - `match()` (*fparser.one.statements.Write method*), 80
  - `match()` (*fparser.one.typedcl\_statements.Byte method*), 108
  - `match()` (*fparser.one.typedcl\_statements.Character method*), 107
  - `match()` (*fparser.one.typedcl\_statements.Class method*), 108
  - `match()` (*fparser.one.typedcl\_statements.Complex method*), 107
  - `match()` (*fparser.one.typedcl\_statements.DoubleComplex method*), 107
  - `match()` (*fparser.one.typedcl\_statements.DoublePrecision method*), 106
  - `match()` (*fparser.one.typedcl\_statements.Implicit method*), 109
  - `match()` (*fparser.one.typedcl\_statements.Integer method*), 106
  - `match()` (*fparser.one.typedcl\_statements.Logical method*), 108
  - `match()` (*fparser.one.typedcl\_statements.Real method*), 106
  - `module`
    - `fparser`, 3
    - `fparser.api`, 3
    - `fparser.common`, 3
    - `fparser.common.base_classes`, 3
    - `fparser.common.readfortran`, 7
    - `fparser.common.sourceinfo`, 13
    - `fparser.common.splitline`, 13
    - `fparser.common.tests`, 14
    - `fparser.common.tests.logging_utils`, 14
    - `fparser.common.utils`, 14
    - `fparser.one`, 17
    - `fparser.one.block_statements`, 17
    - `fparser.one.parsefortran`, 71
    - `fparser.one.statements`, 72
    - `fparser.one.tests`, 105
    - `fparser.one.tests.test_scripts`, 105
    - `fparser.one.typedcl_statements`, 105
    - `fparser.scripts`, 110
    - `fparser.scripts.fparser2`, 110
    - `fparser.scripts.fparser2_bench`, 110
    - `fparser.scripts.parse`, 110
    - `fparser.scripts.read`, 110
    - `fparser.scripts.script_options`, 110
    - `fparser.two`, 111
    - `fparser.two.C99Preprocessor`, 111
    - `fparser.two.Fortran2003`, 111
    - `fparser.two.Fortran2008`, 111
    - `fparser.two.parser`, 111
    - `fparser.two.pattern_tools`, 111
    - `fparser.two.symbol_table`, 111
    - `fparser.two.utils`, 116
  - `Module` (*class in fparser.one.block\_statements*), 21
  - `ModuleProcedure` (*class in fparser.one.block\_statements*), 45
  - `ModuleProcedure` (*class in fparser.one.statements*), 82
  - `MultiLine` (*class in fparser.common.readfortran*), 11
- ## N
- `name` (*fparser.two.symbol\_table.SymbolTable property*), 115
  - `name` (*fparser.two.symbol\_table.SymbolTable.Symbol property*), 114
  - `Namelist` (*class in fparser.one.block\_statements*), 56
  - `Namelist` (*class in fparser.one.statements*), 94
  - `Nullify` (*class in fparser.one.block\_statements*), 50
  - `Nullify` (*class in fparser.one.statements*), 88
- ## O
- `Open` (*class in fparser.one.block\_statements*), 48
  - `Open` (*class in fparser.one.statements*), 85
  - `Optional` (*class in fparser.one.block\_statements*), 57
  - `Optional` (*class in fparser.one.statements*), 95
- ## P
- `Parameter` (*class in fparser.one.block\_statements*), 52
  - `Parameter` (*class in fparser.one.statements*), 89
  - `parent` (*fparser.two.symbol\_table.SymbolTable property*), 115
  - `parse()` (*fparser.one.parsefortran.FortranParser method*), 71
  - `parse_array_spec()` (*in module fparser.common.utils*), 15
  - `parse_bind()` (*in module fparser.common.utils*), 15
  - `parse_result()` (*in module fparser.common.utils*), 15

- ParseError, 16  
 Pause (class in *fparser.one.block\_statements*), 66  
 Pause (class in *fparser.one.statements*), 104  
 Pointer (class in *fparser.one.block\_statements*), 53  
 Pointer (class in *fparser.one.statements*), 91  
 PointerAssignment (class in *fparser.one.block\_statements*), 37  
 PointerAssignment (class in *fparser.one.statements*), 75  
 populate\_use\_provides() (*fparser.one.block\_statements.Use* method), 51  
 populate\_use\_provides() (*fparser.one.statements.Use* method), 89  
 primitive\_type (*fparser.two.symbol\_table.SymbolTable* property), 114  
 Print (class in *fparser.one.block\_statements*), 41  
 Print (class in *fparser.one.statements*), 78  
 Private (class in *fparser.one.block\_statements*), 46  
 Private (class in *fparser.one.statements*), 83  
 process\_item() (*fparser.common.base\_classes.BeginStatement* method), 5  
 process\_item() (*fparser.one.block\_statements.Allocate* method), 44  
 process\_item() (*fparser.one.block\_statements.Associate* method), 31  
 process\_item() (*fparser.one.block\_statements.BeginSource* method), 21  
 process\_item() (*fparser.one.block\_statements.BlockData* method), 25  
 process\_item() (*fparser.one.block\_statements.Call* method), 39  
 process\_item() (*fparser.one.block\_statements.Case* method), 62  
 process\_item() (*fparser.one.block\_statements.ClassIs* method), 63  
 process\_item() (*fparser.one.block\_statements.Do* method), 30  
 process\_item() (*fparser.one.block\_statements.EndDo* method), 36  
 process\_item() (*fparser.one.block\_statements.Enum* method), 32  
 process\_item() (*fparser.one.block\_statements.If* method), 30  
 process\_item() (*fparser.one.block\_statements.IfThen* method), 29  
 process\_item() (*fparser.one.block\_statements.Interface* method), 26  
 process\_item() (*fparser.one.block\_statements.Module* method), 22  
 process\_item() (*fparser.one.block\_statements.Program* method), 24  
 process\_item() (*fparser.one.block\_statements.PythonModule* method), 23  
 process\_item() (*fparser.one.block\_statements.TypeIs* method), 63  
 process\_item() (*fparser.one.block\_statements.Use* method), 51  
 process\_item() (*fparser.one.statements.Allocate* method), 82  
 process\_item() (*fparser.one.statements.Call* method), 76  
 process\_item() (*fparser.one.statements.Case* method), 100  
 process\_item() (*fparser.one.statements.ClassIs* method), 101  
 process\_item() (*fparser.one.statements.TypeIs* method), 100  
 process\_item() (*fparser.one.statements.Use* method), 89  
 process\_subitem() (*fparser.common.base\_classes.BeginStatement* method), 5  
 process\_subitem() (*fparser.one.block\_statements.BeginSource* method), 21  
 process\_subitem() (*fparser.one.block\_statements.Do* method), 30  
 Program (class in *fparser.one.block\_statements*), 23  
 ProgramBlock (class in *fparser.common.base\_classes*), 6  
 Protected (class in *fparser.one.block\_statements*), 53  
 Protected (class in *fparser.one.statements*), 91  
 Public (class in *fparser.one.block\_statements*), 46  
 Public (class in *fparser.one.statements*), 83  
 put\_item() (*fparser.one.parsefortran.FortranParser* method), 72  
 PythonModule (class in *fparser.one.block\_statements*), 22
- ## R
- Read (class in *fparser.one.block\_statements*), 41  
 Read (class in *fparser.one.statements*), 79  
 Read0 (class in *fparser.one.block\_statements*), 42  
 Read0 (class in *fparser.one.statements*), 79  
 Read1 (class in *fparser.one.block\_statements*), 42  
 Read1 (class in *fparser.one.statements*), 79  
 Real (class in *fparser.one.block\_statements*), 67  
 Real (class in *fparser.one.typedecl\_statements*), 106  
 remove() (*fparser.two.symbol\_table.SymbolTables* method), 113  
 Return (class in *fparser.one.block\_statements*), 40  
 Return (class in *fparser.one.statements*), 77  
 Rewind (class in *fparser.one.block\_statements*), 48  
 Rewind (class in *fparser.one.statements*), 85  
 root (*fparser.two.symbol\_table.SymbolTable* property), 115
- ## S
- Save (class in *fparser.one.block\_statements*), 49  
 Save (class in *fparser.one.statements*), 87



- scoping\_unit\_classes  
(*fparser.two.symbol\_table.SymbolTables* property), 113
- SelectCase (class in *fparser.one.block\_statements*), 28
- SelectType (class in *fparser.one.block\_statements*), 29
- Sequence (class in *fparser.one.block\_statements*), 56
- Sequence (class in *fparser.one.statements*), 93
- set\_parse\_options() (in module *fparser.scripts.script\_options*), 110
- set\_read\_options() (in module *fparser.scripts.script\_options*), 110
- SpecificBinding (class in *fparser.one.block\_statements*), 59
- SpecificBinding (class in *fparser.one.statements*), 97
- specs\_split\_comma() (in module *fparser.common.utils*), 15
- split\_comma() (in module *fparser.common.utils*), 15
- splitparen() (in module *fparser.common.splitline*), 13
- splitquote() (in module *fparser.common.splitline*), 13
- Statement (class in *fparser.common.base\_classes*), 4
- Stop (class in *fparser.one.block\_statements*), 41
- Stop (class in *fparser.one.statements*), 78
- str2stmt() (in module *fparser.common.utils*), 15
- String (class in *fparser.common.splitline*), 13
- string\_replace\_map() (in module *fparser.common.splitline*), 13
- Subroutine (class in *fparser.one.block\_statements*), 26
- SYMBOL\_TABLES (in module *fparser.two.symbol\_table*), 116
- SymbolTable (class in *fparser.two.symbol\_table*), 113
- SymbolTable.Symbol (class in *fparser.two.symbol\_table*), 114
- SymbolTableError, 116
- SymbolTables (class in *fparser.two.symbol\_table*), 112
- SyntaxErrorLine, 12
- SyntaxErrorMultiLine, 12
- ## T
- Target (class in *fparser.one.block\_statements*), 53
- Target (class in *fparser.one.statements*), 90
- Threadsafe (class in *fparser.one.block\_statements*), 64
- Threadsafe (class in *fparser.one.statements*), 102
- tofortran() (*fparser.common.base\_classes.EndStatement* method), 5
- tofortran() (*fparser.one.block\_statements.Allocate* method), 45
- tofortran() (*fparser.one.block\_statements.Call* method), 39
- tofortran() (*fparser.one.block\_statements.Case* method), 62
- tofortran() (*fparser.one.block\_statements.ClassIs* method), 63
- tofortran() (*fparser.one.block\_statements.TypeIs* method), 63
- tofortran() (*fparser.one.block\_statements.Use* method), 51
- tofortran() (*fparser.one.statements.Allocate* method), 82
- tofortran() (*fparser.one.statements.Call* method), 76
- tofortran() (*fparser.one.statements.Case* method), 100
- tofortran() (*fparser.one.statements.ClassIs* method), 101
- tofortran() (*fparser.one.statements.TypeIs* method), 100
- tofortran() (*fparser.one.statements.Use* method), 89
- topyf() (*fparser.one.block\_statements.Interface* method), 26
- topyf() (*fparser.one.block\_statements.Module* method), 22
- TypeDecl (in module *fparser.one.block\_statements*), 31
- TypeIs (class in *fparser.one.block\_statements*), 62
- TypeIs (class in *fparser.one.statements*), 100
- TypeStmt (in module *fparser.one.block\_statements*), 69
- TypeStmt (in module *fparser.one.typedecl\_statements*), 108
- ## U
- Use (class in *fparser.one.block\_statements*), 51
- Use (class in *fparser.one.statements*), 88
- ## V
- Value (class in *fparser.one.block\_statements*), 54
- Value (class in *fparser.one.statements*), 92
- Variable (class in *fparser.common.base\_classes*), 5
- Volatile (class in *fparser.one.block\_statements*), 54
- Volatile (class in *fparser.one.statements*), 92
- ## W
- Wait (class in *fparser.one.block\_statements*), 43
- Wait (class in *fparser.one.statements*), 80
- WhereConstruct (in module *fparser.one.block\_statements*), 29
- WhereStmt (in module *fparser.one.block\_statements*), 63
- WhereStmt (in module *fparser.one.statements*), 101
- Write (class in *fparser.one.block\_statements*), 42
- Write (class in *fparser.one.statements*), 80