F2x Documentation

Release 0.2.dev1

Michael Meinel

Aug 01, 2019

CONTENTS

F2x is a Python tool that allows you to take your Fortran code and make it accessible from other languages. Compared to the popular tool f2py it comes with two important differences:

- A superior Fortran parser based on the work by the OpenFortranParser
- A very flexible code generation backend that uses Jinja2 templates

CHAPTER

ONE

REQUIREMENTS

F2x currently requires Python 3 to work. It relies on the following pacakges available from PyPI:

- pyplus
- Jinja2
- numpy
- Cython (optional)

Additional requirements for building documentation and running tests exists. Please consult the *setup.py* for further details. All requirements should be automatically provided during installation of **F2x**.

CHAPTER

TWO

QUICK STEPS

2.1 Getting Started with F2x

This document gives a short introduction to get started working with **F2x**. More detailed information about the single steps can be found in the respective detailed chapter.

2.1.1 Installation

F2x currently depends on Python and *setuptools* in order to install it. If you have these pre-requirements in place, you can go ahead and install **F2x** by cloning the repository and running the setup script:

```
$ git clone https://github.com/DLR-SC/F2x.git
$ cd F2x
$ python setup.py install
```

Of course you need a working *Fortran compiler*. As F2x relies on the build chain provided by numpy, this usually mean that if you can build numpy extensions, you should also be able to build F2x extensions.

This should install all dependencies and a command line tool `F2x` to wrap your Fortran code.

2.1.2 One-shot building

A `tiny example` is available to try F2x. Extract it to some location and from the containing folder run:

```
$ F2x -W lib -m mylib.* mylib/test.f90
```

This applies a *strategy* to generate the wrapper modules and compile the extension named *mylib.test* in one shot. You can try your results by running tests against it:

```
$ python -m mylib.test
          42
```

2.1.3 Wrapping Fortran Sources

If you have successfully installed **F2x**, you can use the `F2x` command line tool to wrap your source files. The general synopsis is:

```
$ F2x [-t template]... [source file]...
```

This will make **F2x** parse the source files and apply each template to each source file to generate the wrapper output. A full overview of all options is available *here* <*command_line*>.

The following templates are the recommended choice:

bindc	`@bindc/_glue.f90.t`	Generate a ISO C ule using BIND(
cerr	`@cerr/_cerr.c.t`	Generates a thin shot for longjmp
ctypes	`@ctypes/_glue.py.t`	Generates a Pyth interface generat cluding error han

Alternatively, you can also use the following set of templates that generates a wrapper without *error handling*.

bindc	`@bindc/_glue.f90.t`	Generate a ISO C ule using BIND(
ctypes_noerr	`@ctypes_noerr/_glue.py.t`	Generates a Pyth interface generat

Usually, you need to apply several templates to achieve full wrapping of your Fortran source (and thus make it usable from Python). You can pass all of them at one to **F2x**.

2.1.4 Building the Extension

After wrapping your Fortran sources, you need to compile the genrated sources. Some of the templates require additional libraries to be used. To include them in compilation, you can use to following helpful commands that returns the full pathes to all required artifacts:

```
$ export F2X_TEMPLATE_LIBS = \
    $(F2x -t bindc -t cerr -t ctypes --get libraries)
$ export F2X_TEMPLATE_MODS = \
    $(F2x -t bindc -t cerr -t ctypes --get modules)
```

2.1.5 Example

Consider the following example Fortran source:

Listing 1: `mylib/test.f90`

```
! Example module to use with F2x documentation.
MODULE TEST

  PUBLIC

CONTAINS

  SUBROUTINE CALL_TEST(INTARG)
     INTEGER, INTENT(IN) :: INTARG

     WRITE (*, *) INTARG
  END SUBROUTINE
```

(continues on next page)

(continued from previous page)

```
13
14 END
```

This file will be wrapped and the resulting sources are compiled to a dynamic library:

```
$ F2x -t @bindc/_glue.f90.t -t @ctypes_noerr/_glue.py.t mylib/test.f90
$ gfortran -fPIC -shared -o mylib/$(F2x --get extlib mylib/test.f90) \
    $(F2x -t bindc -t ctypes_noerr --get libraries) mylib/test.f90 mylib/test_glue.f90
$ cp $(F2x -t bindc -t ctypes_noerr --get modules) mylib
```

Now you should be able to call `CALL_TEST` from Python:

```
$ python
>>> from mylib import test_glue as test
>>> test.CALL_TEST(123)
```

2.1.6 Advanced Options

There are several other ways to interact with **F2x** which are described in more detail in their respective sections.

- *Using setup.py to build extensions*
- *Tuning interface generation parameters for a Fortran source*
- *Extending F2x with your own templates*
- *Using an alternate parser for F2x*

2.2 Using F2x from `setup.py`

F2x comes with a very good support for *distutils* based on the great work by *numpy <http://numpy.org>*. Thanks to own implementations of *`Extension`*, `build_src`, and `build_ext`.

To take advantage of the adopted build processes, you simply need to use the correct imports:

```
from F2x.distutils import setup, Extension
```

There is one additional required parameter to be added to your *`Extension`*. You need to provide a wrapping strategy for your extension. The following strategies are already available:

lib	*@bindc/_glue.f90.t*, *@cerr/_cerr.c.t*, *@ctypes/_glue.py.t*	*not documented*
lib_noerr	*_glue.f90.t*, *_glue.py.t*	*not documented*

The chapter *Build Strategies* explains build strategies in more detail.

2.2.1 Including own templates

You can use your own templates by adding them to the template registry. Your template needs to be contained in a *F2x template package*. Then you can simply add that package to the registry:

```
from F2x.template import register_template
import my_template_package

register_template(my_template_package)
```

The allows to reference the template by its name from a custom strategies or the *Extensions* definitions using the `templates` attribute.

2.3 Trouble Shooting

CHAPTER

THREE

USER MANUAL

3.1 User Manual

3.1.1 Command Line Tool

F2x - A versatile Fortran wrapper

```
usage: F2x [-h] [-c CONFIG] [-G GRAMMAR] [-C CONFIG_SUFFIX] [-P] [-F]
           [-e ENCODING] [-i TREE_CLASS] [-W STRATEGY] [-m MODULE_NAME]
           [-n LIBRARY_NAME] [-s] [-S NAME CLASS TEMPLATES] [-R PACKAGE]
           [-t NAME] [-T PATH] [-x EXTENSION] [-l LOGFILE] [-v] [-q] [-f]
           [--get {depends,modules,libraries,extlib}] [-d]
           [--py-absolute-import]
           [SOURCE [SOURCE ...]]
```

Named Arguments

-c, --config Load configuration file.

Fortran parser

-G, --grammar Use specified grammar. Bundled grammars should be prefixed by @. (Default: "@fortran.g")

Default: "@fortran.g"

-C, --config-suffix Suffix for per-source configuration file. (Default: "-wrap")

Default: "-wrap"

-P, --output-pre Write pre-processed source.

Default: False

-F, --configure Create/update configuration file.

Default: False

-e, --encoding Use the specified encoding for reading/writing source files.

Default: "utf8"

-i, --tree-class Tree class to use for parsing (module.name:ClassName).

Automatic wrapping

-W, --wrap Wrap sources by applying the given STRATEGY.

-m, --module-name Full name of the module to generate.

Default: "ext.*"

-n, --library-name Set the library name.

-s, --autosplit Automatically create an own extension for each source file.

Default: False

-S, --add-strategy Load a strategy for later use. NAME should be the name for the strategy and CLASS should be the fully qualified class name of a build strategy. TEMPLATES is a comma separated list of templates to use.

Code generation

-R, --register-template Load a template pacakge into the regitry for later use. PACAKGE should be the fully qualified name of a F2x template package.

-t, --template Generate wrapping for the given template. NAME should be the name of a F2x template package, the path to a template that can be found in the template path or a bundled template that should be prefixed by @.

Default: []

-T, --template-path Add PATH to the template search path.

Default: []

-x, --jinja-ext Add EXTENSION to Jinja2 environment.

Default: ['jinja2.ext.do']

Logging

-l, --logfile Write detailed log to LOGFILE.

-v, --verbose Increase verbosity.

Default: 0

-q, --quiet Decrease verbosity.

Default: 2

Action

-f, --force Force rebuild of wrapper.

Default: False

--get Possible choices: depends, modules, libraries, extlib

Collect template information about dependencies, modules, libraries, or the name of the extension library.

SOURCE Wrap the given templates to the SOURCE files.

Deprecated

-d, --copy-glue Copy 'glue.py' (used by ctypes template) into destination folder.

Default: False

--py-absolute-import Use absolute import for Python helper modules (e.g. F2x.template.ctypes.glue).

Default: False

3.1.2 Interface Configuration File

You can put a `source.f90-wrap` file along your `source.f90` to further specify the interface to be exported.

3.1.3 Compilers

F2x needs a working set of compilers to be fully functioning. This means you need to have compatible Fortran and C compilers at hand. F2x relies on the build system of numpy ant its build support. If you are able to build numpy extensions with your environment, you should be fine.

The main target of the development focusses on the Intel Fortran and GFortran compilers.

3.1.4 Templates

F2x uses templates to generate the code. There are a bunch of templates that come bundled with **F2x**:

bindc	`@bindc/_glue.f90.t`	Generate a ISO C ule using BIND(
cerr	`@cerr/_cerr.c.t`	Generates a thin C shot for longjmp
ctypes	`@ctypes/_glue.py.t`	Generates a Pyth interface generat cluding error han
ctypes_noerr	`@ctypes_noerr/_glue.py.t`	Generates a Pyth interface generat
sphinx	`.rst.t`	Generates a Sphi

3.1.5 Choosing a Template on Command Line

When you use the *F2x CLI* to wrap your sources, you can select templates by specifying the *-t* switch. You have three possibilities of accessing template files:

- The name of a built-in template. These start with an '@' character and are listed in the table above.
- The full (relative) path to a template file.
- A path relative to one of the template path directories. You can adjust the template path using the *-T* switch.

3.1.6 Build Strategies

F2x extends the numpy build system using strategies that interact with the build environment at defined points during the build process. This allows to decouple specific tweaks that are done from the overall build process. In general,

every set of templates needs a specific build strategy. In return, every build strategy usally has it's specific set of templates that it needs to successfully build a Python extension.

The base `BuildStrategy` defines the seqence when it can interact with the build process.

Available Strategies

A set of strategies come bundled with F2x:

lib	*@bindc/_glue.f90.t*, *@cerr/_cerr.c.t*, *@ctypes/_glue.py.t*	*not documented*
lib_noerr	*_glue.f90.t*, *_glue.py.t*	*not documented*

Error Handling

Ususally, error handling should be done using return values etc. However, in many cases Fortran programs seem to simply `STOP` running if they face a condition. This is a big show stopper for Python as the whole process will be killed. To accomodate this problem, F2x implements a thin C wrapper using *F2x.template.cerr* that qualifies as `longjmp` traget. This allows us to replace all the calls to `STOP` by a calls to *F2X_ERR_HANDLE()*. This triggers the `longjmp` and program flow returns to Python.

To support this method, you need to make sure to use the correct strategy or templates:

Strategies with error handling:

lib	*@bindc/_glue.f90.t*, *@cerr/_cerr.c.t*, *@ctypes/_glue.py.t*	*not documented*

Templates with error handling:

bindc	`@bindc/_glue.f90.t`	Generate a ISO C ule using BIND(
cerr	`@cerr/_cerr.c.t`	Generates a thin C shot for longjmp
ctypes	`@ctypes/_glue.py.t`	Generates a Pyth interface generat cluding error han

3.2 Advanced Topics

3.2.1 Using Custom Templates

3.2.2 Using an Alternate Parser

not supported yet

3.3 Publications

Some papers have been written about F2x so far

- Michael Meinel: F2x - a versatile wrapper generator, EuroSciPy 2018, Trento/Italy

3.4 Table of Contents

3.4.1 Introduction

3.4.2 F2x package

This file provides information about the version of F2x you are using.

F2x.**get_version_string**(*full=False*)

Subpackages

F2x.distutils package

Subpackages

F2x.distutils.command package

Submodules

F2x.distutils.command.build_ext module

F2x implementation of the *build_ext* command for distutils (*setup.py*).

This implementation basically inserts some interaction points. Namely, it calls *prepare_build_extension*, *finish_build_extension*, and *get_ext_filename*. It also ensures that a build strategy is available.

See also:

F2x.distutils.strategy.base Details about the build process and build strategies are documented in the documentation of the base *BuildStrategy*.

class F2x.distutils.command.build_ext.**build_ext**(*dist*)
Bases: numpy.distutils.command.build_ext.build_ext

build_extension(*ext*)

finalize_options()
Set final values for all the options that this command supports. This is always called as late as possible, ie. after any option assignments from the command-line or from other commands have been done. Thus, this is the place to code option dependencies: if 'foo' depends on 'bar', then it is safe to set 'foo' from 'bar' as long as 'foo' still has the same value it was assigned in 'initialize_options()'.

This method must be implemented by all command classes.

get_ext_filename(*ext_name*)
Convert the name of an extension (eg. "foo.bar") into the name of the file from which it will be loaded (eg. "foo/bar.so", or "foobar.pyd").

initialize_options()
Set default values for all the options that this command supports. Note that these defaults may be overridden by other commands, by the setup script, by config files, or by the command-line. Thus, this is not the place to code dependencies between options; generally, 'initialize_options()' implementations are just a bunch of "self.foo = None" assignments.

This method must be implemented by all command classes.

F2x.distutils.command.build_ext.**get_strategy**()
D.get(k[,d]) -> D[k] if k in D, else d. d defaults to None.

F2x.distutils.command.build_sphinx module

class F2x.distutils.command.build_sphinx.**build_sphinx**(*dist*)
Bases: sphinx.setup_command.BuildDoc

Build documentation based on Sphinx.

This implementation adds automatic generation of the API documentation (`sphinx-apidoc ...`) during the build.

run()
A command's raison d'etre: carry out the action it exists to perform, controlled by the options initialized in 'initialize_options()', customized by other commands, the setup script, the command-line, and config files, and finalized in 'finalize_options()'. All terminal output and filesystem interaction should be done by 'run()'.

This method must be implemented by all command classes.

use_f2x_template = False

F2x.distutils.command.build_sphinx.**document_templates**(*source_dir*)
Write a Sphinx documentation file for each template (including dependencies).

Parameters **source_dir** – Base directory to write output to.

F2x.distutils.command.build_src module

class F2x.distutils.command.build_src.**build_src**(*dist*)
Bases: numpy.distutils.command.build_src.build_src

Build sources for an F2x extension.

This module creates source files for an extension. It proceeds by applying F2x with a given set of templates on the (appropriate) sources. For transformation of the sources, a given `BuildStrategy` may be applied.

boolean_options = ['force', 'inplace']

build_sources()

description = 'build sources from F2x'

finalize_options()
Set final values for all the options that this command supports. This is always called as late as possible, ie. after any option assignments from the command-line or from other commands have been done. Thus, this is the place to code option dependencies: if 'foo' depends on 'bar', then it is safe to set 'foo' from 'bar' as long as 'foo' still has the same value it was assigned in 'initialize_options()'.

This method must be implemented by all command classes.

get_target_dir(*extension*)

help_options = [('help-strategies', None, 'list available strategies', <function show_

initialize_options()
Set default values for all the options that this command supports. Note that these defaults may be overridden by other commands, by the setup script, by config files, or by the command-line. Thus, this is not the place to code dependencies between options; generally, 'initialize_options()' implementations are just a bunch of "self.foo = None" assignments.

This method must be implemented by all command classes.

populate_build_src(*extension*)

prepare_package(*package_name*)

select_sources(*extension*, *strategy*, *target_dir*, *sources_to_wrap*)

user_options = [('build-src=', 'd', 'directory to "build" sources to'), ('strategy=', 1

wrap_sources(*sources_to_wrap*)

F2x.distutils.command.build_src.**get_strategy**()
D.get(k[,d]) -> D[k] if k in D, else d. d defaults to None.

F2x.distutils.strategy package

This module controls the built-in build strategies. It provides a registry that can be used to add and retrieve custom build strategies.

See also:

F2x.distutils.strategy.base.BuildStrategy The documentation of *F2x.distutils.strategy.base.BuildStrategy* contains details about the build process and how to modify it with own build strategies.

F2x.distutils.strategy.library.ExtensionLibBuildStrategy A build strategy to create Python extensions that need to load a library with the compiled wrapper code (like the *F2x.template.ctypes* template).

F2x.distutils.strategy.extension.ExtensionBuildStrategy A build strategy to create Python C extensions that contain the wrapper code in a loadable module.

F2x.distutils.strategy.**get_strategy**()
D.get(k[,d]) -> D[k] if k in D, else d. d defaults to None.

F2x.distutils.strategy.**register_strategy**(*name*, *strategy*)
Add a new strategy to the registry. If a strategy with the same name is already registered, it will be overwritten.

Parameters

- **name** – Name for the new strategy.
- **strategy** – An instance of a subclass of BuildStrategy.

F2x.distutils.strategy.**show_strategies**()

Submodules

F2x.distutils.strategy.base module

class F2x.distutils.strategy.base.**BuildStrategy**(*templates=None*)
Bases: object

Basic build strategy.

A build strategy follows the build process and interacts with it at certain well-defined points.

1. build_src - Generate and prepare sources for compilation.
    1. *prepare_distribution()*
    2. *prepare_extension()* for each *Extension*
    3. individually wrap each *Extension*
        1. *prepare_wrap_sources()*
        2. wrap sources from *select_wrap_sources()*
        3. *finish_wrap_sources()*
    4. *finish_distribution()*
2. build_ext - Actually build binaries from (generated) sources.
    1. individually build each *Extension*
        1. *prepare_build_extension()*
        2. compile extension to *get_ext_filename()*
        3. *finish_build_extension()*

Where each strategy intervenes and what it does is documented in the implementations.

finish_build_extension(*build_ext*, *extension*)
No customization here.

finish_distribution(*build_src*, *distribution*)
Update *build_clib* and *build_py* steps with newly collected libraries.

finish_wrap_sources(*build_src*, *extension*, *target_dir*)
Clean up after code generation. Put newly generated files where they belong.

Also creates a new library for the primary sources if requested by inline_sources.

get_ext_filename(*build_src*, *extension*)
No customization here (i.e., return None).

get_template_files(*with_imports=False*)
Collect all template files required for this strategy.

Parameters **with_imports** – If set to True this will also try to extract imports for the the templates recursively.

load_templates(*template_names*)
Load a list with template names into a list with templates and extra information:

- loaded template
- template file name (as passed to loader)
- full path to loaded template
- package directory of containing package

prepare_build_extension(*build_ext*, *extension*)
Prepare build by updating include directories.

prepare_distribution(*build_src*, *distribution*)
Make sure *distribution.libraries* is at least an empty list.

prepare_extension(*build_src*, *extension*)
Prepare extension for code generation.

- Collect information about extension sources into `ext_modules`.
- Decide whether to split and split.
- Collect libraries and modules form templates.

prepare_wrap_sources(*build_src*, *extension*, *target_dir*)
Prepare sources for wrapping. Make sure everything is where it is expected.

select_wrap_sources(*build_src*, *extension*, *target_dir*)
Collect information about sources to be built.

This method collects the following information about the module surces and passes them on:

- name of original source file
- target name of source file (from where code generation will take place)
- names of new files to be expected
- dependencies of those new files (i.e., templates, sources, interface config)

F2x.distutils.strategy.base.**f90_module_name_match**()
Matches zero or more characters at the beginning of the string.

F2x.distutils.strategy.base.**fortran_ext_match**()
Matches zero or more characters at the beginning of the string.

F2x.distutils.strategy.extension module

class F2x.distutils.strategy.extension.**ExtensionBuildStrategy**(*templates=None*)
Bases: *F2x.distutils.strategy.base.BuildStrategy*

F2x.distutils.strategy.extension.**c_ext_match**()
Matches zero or more characters at the beginning of the string.

F2x.distutils.strategy.extension.**cxx_ext_match**()
Matches zero or more characters at the beginning of the string.

F2x.distutils.strategy.extension.**fortran_ext_match**()
Matches zero or more characters at the beginning of the string.

F2x.distutils.strategy.library module

class F2x.distutils.strategy.library.**ExtensionLibBuildStrategy**(*templates=None*)
Bases: *F2x.distutils.strategy.extension.ExtensionBuildStrategy*

finish_build_extension(*build_ext*, *extension*)
No customization here.

get_ext_filename(*build_src*, *ext_name*)
No customization here (i.e., return None).

prepare_build_extension (*build_ext*, *extension*)
Prepare build by updating include directories.

prepare_extension (*build_src*, *extension*)
Prepare extension for code generation.

- Collect information about extension sources into `ext_modules`.
- Decide whether to split and split.
- Collect libraries and modules form templates.

prepare_wrap_sources (*build_src*, *extension*, *target_dir*)
Prepare sources for wrapping. Make sure everything is where it is expected.

select_wrap_sources (*build_src*, *extension*, *target_dir*)
Collect information about sources to be built.

This method collects the following information about the module surces and passes them on:

- name of original source file
- target name of source file (from where code generation will take place)
- names of new files to be expected
- dependencies of those new files (i.e., templates, sources, interface config)

Submodules

F2x.distutils.core module

F2x.distutils.extension module

class `F2x.distutils.extension.`**Extension** (*name*, *sources*, ***kwargs*)
Bases: `numpy.distutils.extension.Extension`

clone (*name*, *sources=None*)
Duplicate this extension.

The new extension will get a new name and might also get a new set of sources. The `autosplit` flag is reset to `None` to avoid infinite splits.

Parameters **name** – The name for the new extension.

copy_to (*other*)

F2x.parser package

Subpackages

F2x.parser.plyplus package

Subpackages

F2x.parser.plyplus.grammar package

Submodules

F2x.parser.plyplus.source module

Created on 12.02.2016

@author: meinel

class F2x.parser.plyplus.source.**SourceFile**(*filename*, *args*)
Bases: *F2x.parser.source.SourceFile*

get_gtree(*cls=None*)

parse()

F2x.parser.plyplus.source.**load_grammar**(*grammar_filename*)

F2x.parser.plyplus.tree module

Created on 08.04.2016

@author: meinel

class F2x.parser.plyplus.tree.**FuncDef**(*ast*)
Bases: *F2x.parser.plyplus.tree.SubDef*

class F2x.parser.plyplus.tree.**Module**(*ast*)
Bases: *F2x.parser.tree.Module*

export_methods(*src*)

class F2x.parser.plyplus.tree.**SubDef**(*ast*)
Bases: *F2x.parser.tree.SubDef*

class F2x.parser.plyplus.tree.**TypeDef**(*ast*)
Bases: *F2x.parser.tree.TypeDef*

class F2x.parser.plyplus.tree.**VarDecl**(*ast*, *prefix=''*)
Bases: *F2x.parser.tree.VarDecl*

A variable declaration.

The following properties are available:

- name: The symbolic name of the variable.
- **type: The C type of this variable. This might be a basic type (REAL, INTEGER, LOGICAL) or TYPE(C) for any** other type like arrays, derived types or strings.
- pytype, cstype: The type to be used by Python or C# respectively.
- intent: May be 'IN', 'OUT' or 'INOUT'.
- getter: This indicates whether the generated getter should be a 'function' or 'subroutine'.
- setter (opt): This indicates whether a 'subroutine' should be generated as setter.
- ftype (opt): The name of the derived type.
- strlen (opt): The length of the string.
- kind (opt): The kind specifier if available.
- dynamic (opt): Indicates whether the variable is 'ALLOCATABLE' or a 'POINTER'.

- dims (opt): For an array contains a list with the sizes per dimension.

with_intent(*intent*)

Submodules

F2x.parser.source module

Created on 12.02.2016

@author: meinel

class F2x.parser.source.**SourceFile**(*filename*, *args*)
Bases: object

get_gtree()

parse()

preprocess(*rules=None*)

read()

F2x.parser.source.**load_grammar**(*grammar_filename*)

F2x.parser.tree module

This module contains the base classes for the *Abtract Generation Tree* that is built by the parser form the Fortran sources.

class F2x.parser.tree.**FuncDef**(*ast*)
Bases: *F2x.parser.tree.SubDef*

class F2x.parser.tree.**Module**(*ast*)
Bases: *F2x.parser.tree.Node*

class F2x.parser.tree.**Node**(*ast*)
Bases: dict

Node constructor stores local AST node in _ast and calls _init_children() which should be overwritten by child classes.

This is the base class for the simplified AST that can easily be used in templates. It is simply a dict which stores child nodes as values. This allows to simply use node.child to access the values from a template. E.g. to get the modules name, you can simply use

```
{{ module.name }}
```

class F2x.parser.tree.**SubDef**(*ast*)
Bases: *F2x.parser.tree.Node*

class F2x.parser.tree.**TypeDef**(*ast*)
Bases: *F2x.parser.tree.Node*

class F2x.parser.tree.**VarDecl**(*ast*, *prefix=''*)
Bases: *F2x.parser.tree.Node*

A variable declaration.

The following properties are available:

name		The symbolic name of the variable.
type		The C type of this variable. This might be a basic type (REAL, INTEGER, LOGICAL) or TYPE(C) for any other type like arrays, derived types or strings.
intent		May be 'IN', 'OUT', or 'INOUT'.
getter		This indicates whether the generated getter should be a FUNCTIN or SUBROUTINE'.
setter	(opt)	This indicates whether a SUBROUTINE should be generated as setter.
ftype	(opt)	The name of the derived type.
strlen	(opt)	The length of the string.
kind	(opt)	The KIND specifier if available.
dynamic	(opt)	Indicates whether the variable is ALLOCATABLE or a POINTER.
dims	(opt)	For an array contains a list with the sizes per dimension.

with_intent(*intent*)

F2x.runtime package

Submodules

F2x.runtime.argp module

F2x.runtime.argp.**get_arg**(*args*, *config*, *name*, *section*, *default*, *cast*)

F2x.runtime.argp.**get_args_parser**()

F2x.runtime.argp.**init_logger**(*args*, *parent=None*, *fmt=None*)

F2x.runtime.argp.**parse_args**(*argv=None*)

F2x.runtime.daemon module

class F2x.runtime.daemon.**F2xClient**(*host*, *port*)

Bases: object

invoke(*args*)

class F2x.runtime.daemon.**F2xDaemon**(*addr*, *port*, *num_procs*)

Bases: object

serve()

class F2x.runtime.daemon.**SocketStream**(*sock*, *mode*)

Bases: socket.SocketIO

write(*data*)

Write the given bytes or bytearray object *b* to the socket and return the number of bytes written. This can be less than len(b) if not all data could be written. If the socket is non-blocking and no bytes could be written None is returned.

F2x.runtime.daemon.**main**()

F2x.runtime.main module

Main program for F2x - A versatile, template based FORTRAN wrapper.

F2x.runtime.main.**main**(*argv=None*, *from_distutils=False*)

F2x.runtime.wrapper module

class F2x.runtime.wrapper.**F2xWrapper**(*args*, *log*)
Bases: object

run()

F2x.runtime.wrapper.**get_strategy**()
D.get(k[,d]) -> D[k] if k in D, else d. d defaults to None.

F2x.template package

F2x.template.**collect_template_data**(*include=None*)
Collect all data files that are required by any registered template.

This includes:

- all template files and their dependencies
- all library files included in the templates
- all Python modules included in the templates

Parameters **include** – A set of strings that indicates what to collect. Possible choices are 'templates', 'depends', 'libraries', and 'modules'. If nothing is select explicitly, everything will be collected.

Returns A generator that yields pairs of template and the requested data files.

F2x.template.**get_library_sources**(**templates*)
Collect source files for all libraries in the given list of templates.

Parameters **templates** – A list of template names or template modules.

Returns A list with all library files with full path that are required by the given templates.

F2x.template.**get_template**(*name: str*) → module
Retrieve a loaded template from the registry.

Parameters **name** – Name of a loaded template.

Returns The template module.

F2x.template.**register_template**(*template*)
Add a new template to the registry.

Before the module is registered, some preprocessing is made:

- **Check if all required attributes of the template are set. These are:**
  - *name*: The name of the template. This will later be used to reference the template.
  - *templates*: A list of templates that should be rendered.
  - *requires*: Other templates that need to be rendered in order for this template to work. May be *None*.
  - *modules*: A list of Python modules the rendered output requires to work.

  - ***libraries*: A list of libraries the compiled output of this template needs to work. You may** use only module names if the required libraries are added to the distribution in any other way. Otherwise use a tuple with the library name and a library spec dict like used for numpy.
  - A docstring.
- **The following attributes are added:**
  - *package_dir*: The directory of the template package.
  - *template_files*: A list with full pathes of all template files to be rendered.
  - *depends*: A list of dependencies for all rendered templates.
- The docstring is extended to reference templates and libraries.

Parameters **template** – The template module to register.

F2x.template.**show_templates**(*out=<_io.TextIOWrapper name='<stdout>' mode='w' encoding='UTF-8'>*, *full_doc=False*)

Print a nicely formatted list of templates and their documentation.

Parameters **out** – The file to write to.

Subpackages

F2x.template.bindc package

Generate a ISO C compliant interface to a Fortran module using BIND(C).

Templates

_glue.f90.t

TEMPLATE bindc/_glue.f90.t

```
-##############################################################################################################
```

calls.f90.tl

TEMPLATE bindc/calls.f90.tl

```
-##############################################################################################################
```

types.f90.tl

TEMPLATE bindc/types.f90.tl

```
-##############################################################################################################
```

vars.f90.tl

TEMPLATE bindc/vars.f90.tl

```
-#############################################################################################################
```

Libraries

Library bindc_f2x

This library contains utility routines required by the *bindc* template for marshalling data between C and Fortran.

C_INTERFACE_MODULE

F2x.template.bindc_new package

Generate a ISO C compliant interface to a Fortran module using BIND(C).

Templates

_glue.f90.t

TEMPLATE bindc_new/_glue.f90.t

```
-#############################################################################################################
```

types.f90.tl

TEMPLATE bindc_new/types.f90.tl

```
-#############################################################################################################
```

methods.f90.tl

TEMPLATE bindc_new/methods.f90.tl

```
-#############################################################################################################
```

types_vars.f90.tl

TEMPLATE bindc_new/types_vars.f90.tl

```
-#############################################################################################################
```

marshal/names.f90.tl

TEMPLATE bindc_new/marshal/names.f90.tl

- Marshalling names. -

marshal/args.f90.tl

TEMPLATE bindc_new/marshal/args.f90.tl

-##

marshal/types.f90.tl

TEMPLATE bindc_new/marshal/types.f90.tl

- Marshalling types. -

Libraries

Library f2x_bindc

Module F2X_BINDC

subroutine F2X_BINDC/**F2X_SET_ERROR**(*ERROR_CODE*, *ERROR_MESSAGE*)

Set an error code an message.

Parameters **ERROR_CODE** *[INTEGER]* :: The error code to set.

Options **ERROR_MESSAGE** *[CHARACTER(60)]* :: An optional error message.

F2x.template.cerr package

Generates a thin C layer that is used as clean stack snapshot for longjmp error handling.

Templates

_cerr.c.t

TEMPLATE cerr/_cerr.c.t

-##

Libraries

Library cerr_f2x

This library contains helpers that allow error handling by using longjmp from code. This can be used to replace hard exits as an error handling.

F2X_ERR

subroutine F2X_ERR/**F2X_ERR_HANDLE**(*CODE*)
Set an error code and return to Python caller. The Fortran control flow is interrupted.

Parameters **CODE** *[INTEGER]* :: The error code to be set (will be included in Python exception).

f2x_err_impl.c

static jmp_buf **f2x_err_jmp_buf**
Holds the `jmp_buf` for the current call.

static bool **f2x_err_active**
Indicates wheather a call is currently active.

static int **f2x_err_code**
Holds an error code for the last call. Use *f2x_err_get()* to read status and *f2x_err_reset()* to reset it.

jmp_buf ***f2x_prepare_jmp_buffer**()
Prepare *f2x_err_jmp_buf*. If the buffer is already in use, indicate an error by setting *f2x_err_code* to -1.

Returns Address of *f2x_err_jmp_buf* (or 0 if jump buffer is in use).

void **f2x_clear_jmp_buffer**()
Cleanup *f2x_err_jmp_buf* after a call finished and release all resources.

void **f2x_err_handle**(int *code*)
Trigger error handler. This will set the *f2x_err_code* to the given error and use *f2x_err_jmp_buf* to stop the execution of the current call.

f2x_err_jmp_buf will be clean up for next use.

Parameters

- **code** – The error code that should be set.

void **f2x_err_reset**()
Reset *f2x_err_code* to 0 (no error).

int **f2x_err_get**()
Get current value of *f2x_err_code*.

Returns Value of *f2x_err_code*.

F2x.template.ctypes package

Generates a Python module that interacts with a ISO C interface generated by 'bindc' template using ctypes including error handling using 'cerr' template.

Templates

_glue.py.t

TEMPLATE ctypes/_glue.py.t

```
-#################################################################################################################
```

calls.py.tl

TEMPLATE ctypes/calls.py.tl

```
-#################################################################################################################
```

types.py.tl

TEMPLATE ctypes/types.py.tl

```
-#################################################################################################################
```

F2x.template.ctypes_new package

Generates a Python module that interacts with a ISO C interface generated by 'bindc' template using ctypes including error handling using 'cerr' template.

Templates

_glue.py.t

TEMPLATE ctypes_new/_glue.py.t

```
-#################################################################################################################
```

types.py.tl

TEMPLATE ctypes_new/types.py.tl

- Types. -

methods.py.tl

TEMPLATE ctypes_new/methods.py.tl

- Methods. -

marshal/bind.py.tl

TEMPLATE ctypes_new/marshal/bind.py.tl

- Bindings. -

marshal/args.py.tl

TEMPLATE ctypes_new/marshal/args.py.tl

- Marshal arguments -

marshal/names.py.tl

TEMPLATE ctypes_new/marshal/names.py.tl

- Marshal names. -

marshal/types.py.tl

TEMPLATE ctypes_new/marshal/types.py.tl

- Marshal types. -

F2x.template.ctypes_noerr package

Generates a Python module that interacts with a ISO C interface generated by 'bindc' template using ctypes.

Templates

_glue.py.t

TEMPLATE ctypes_noerr/_glue.py.t

```
-#################################################################################################################
```

calls.py.tl

TEMPLATE ctypes_noerr/calls.py.tl

```
-#################################################################################################################
```

types.py.tl

TEMPLATE ctypes_noerr/types.py.tl

```
-#################################################################################################################
```

F2x.template.sphinx package

Generates a Sphinx documentation for a module.

Templates

.rst.t

TEMPLATE sphinx/.rst.t

- Sphinx documentation. -

CHAPTER

FOUR

INDICES AND TABLES

- *Table of Contents*
- *F2x package*
- genindex
- modindex
- search

PYTHON MODULE INDEX

f

FORTRAN MODULE INDEX

TEMPLATES

bindc

bindc_new

bindc_new/marshal

cerr

ctypes

ctypes_new

ctypes_new/marshal

ctypes_noerr

sphinx

INDEX

G

H

I

L

M

N

P

R

S

T

U

V

W