

Using GNU Fortran

For GCC version 17.0.0 (pre-release)

(GCC)

The gfortran team

Published by the Free Software Foundation
51 Franklin Street, Fifth Floor
Boston, MA 02110-1301, USA

Copyright © 1999-2026 Free Software Foundation, Inc.

Permission is granted to copy, distribute and/or modify this document under the terms of the GNU Free Documentation License, Version 1.3 or any later version published by the Free Software Foundation; with the Invariant Sections being “Funding Free Software”, the Front-Cover Texts being (a) (see below), and with the Back-Cover Texts being (b) (see below). A copy of the license is included in the section entitled “GNU Free Documentation License”.

(a) The FSF’s Front-Cover Text is:

A GNU Manual

(b) The FSF’s Back-Cover Text is:

You have freedom to copy and modify this GNU Manual, like GNU software. Copies published by the Free Software Foundation raise funds for GNU development.

Short Contents

1	Introduction	1
	Invoking GNU Fortran	
2	GNU Fortran Command Options	7
3	Runtime: Influencing runtime behavior with environment variables	39
	Language Reference	
4	Compiler Characteristics	45
5	Extensions	51
6	Mixed-Language Programming	75
7	Coarray Programming	91
8	Intrinsic Procedures	121
9	Intrinsic Modules	315
	Contributing	325
	GNU General Public License	327
	GNU Free Documentation License	339
	Funding Free Software	347
	Option Index	349
	Keyword Index	351

Table of Contents

1	Introduction	1
1.1	About GNU Fortran	1
1.2	GNU Fortran and GCC	2
1.3	Standards	3
1.3.1	Fortran 95 status	3
1.3.2	Fortran 2003 status	3
1.3.3	Fortran 2008 status	4
1.3.4	Fortran 2018 status	4
	Part I: Invoking GNU Fortran	5
2	GNU Fortran Command Options	7
2.1	Option summary	7
2.2	Options controlling Fortran dialect	9
2.3	Enable and customize preprocessing	15
2.4	Options to request or suppress errors and warnings	18
2.5	Options for debugging your program	23
2.6	Options for directory search	25
2.7	Influencing the linking step	25
2.8	Influencing runtime behavior	25
2.9	GNU Fortran Developer Options	26
2.10	Options for code generation conventions	27
2.11	Options for interoperability with other languages	35
2.12	Environment variables affecting <code>gfortran</code>	36
2.13	Shared-memory Coarrays	36
3	Runtime: Influencing runtime behavior with environment variables	39
3.1	TMPDIR—Directory for scratch files	39
3.2	GFORTRAN_STDIN_UNIT—Unit number for standard input	39
3.3	GFORTRAN_STDOUT_UNIT—Unit number for standard output	39
3.4	GFORTRAN_STDERR_UNIT—Unit number for standard error	39
3.5	GFORTRAN_UNBUFFERED_ALL—Do not buffer I/O on all units	39
3.6	GFORTRAN_UNBUFFERED_PRECONNECTED—Do not buffer I/O on preconnected units	39
3.7	GFORTRAN_SHOW_LOCUS—Show location for runtime errors	39
3.8	GFORTRAN_OPTIONAL_PLUS—Print leading + where permitted	40
3.9	GFORTRAN_LIST_SEPARATOR—Separator for list output	40
3.10	GFORTRAN_CONVERT_UNIT—Set conversion for unformatted I/O	40
3.11	GFORTRAN_ERROR_BACKTRACE—Show backtrace on run-time errors	41
3.12	GFORTRAN_FORMATTED_BUFFER_SIZE—Set buffer size for formatted I/O	41

3.13	GFORTTRAN_UNFORMATTED_BUFFER_SIZE—Set buffer size for unformatted I/O	41
------	--	----

Part II: Language Reference 43

4 Compiler Characteristics 45

4.1	KIND Type Parameters	45
4.2	Internal representation of LOGICAL variables	45
4.3	Evaluation of logical expressions	46
4.4	MAX and MIN intrinsics with REAL NaN arguments	46
4.5	Thread-safety of the runtime library	46
4.6	Data consistency and durability	47
4.7	Files opened without an explicit ACTION= specifier	48
4.8	File operations on symbolic links	48
4.9	File format of unformatted sequential files	48
4.10	Asynchronous I/O	49
4.11	Behavior on integer overflow	49

5 Extensions 51

5.1	Extensions implemented in GNU Fortran	51
5.1.1	Old-style kind specifications	51
5.1.2	Old-style variable initialization	51
5.1.3	Extensions to namelist	52
5.1.4	X format descriptor without count field	53
5.1.5	Commas in FORMAT specifications	53
5.1.6	Missing period in FORMAT specifications	53
5.1.7	Default widths for 'F', 'G' and 'I' format descriptors	53
5.1.8	I/O item lists	53
5.1.9	'Q' exponent-letter	53
5.1.10	BOZ literal constants	54
5.1.11	Real array indices	54
5.1.12	Unary operators	54
5.1.13	Implicitly convert LOGICAL and INTEGER values	54
5.1.14	Hollerith constants support	54
5.1.15	Character conversion	55
5.1.16	Cray pointers	56
5.1.17	CONVERT specifier	57
5.1.18	OpenMP	58
5.1.19	OpenACC	59
5.1.20	Argument list functions %VAL, %REF and %LOC	59
5.1.21	Read/Write after EOF marker	60
5.1.22	STRUCTURE and RECORD	60
5.1.23	UNION and MAP	63
5.1.24	Type variants for integer intrinsics	64
5.1.25	AUTOMATIC and STATIC attributes	66

5.1.26	Form feed as whitespace	66
5.1.27	TYPE as an alias for PRINT	66
5.1.28	%LOC as an rvalue	67
5.1.29	.XOR. operator	67
5.1.30	Bitwise logical operators	67
5.1.31	Extended I/O specifiers	67
5.1.32	Legacy PARAMETER statements	69
5.1.33	Default exponents	69
5.1.34	Unsigned integers	69
5.2	Extensions not implemented in GNU Fortran	71
5.2.1	ENCODE and DECODE statements	72
5.2.2	Variable FORMAT expressions	72
5.2.3	Alternate complex function syntax	73
5.2.4	Volatile COMMON blocks	73
5.2.5	OPEN(... NAME=)	73
5.2.6	Q edit descriptor	73
6	Mixed-Language Programming	75
6.1	Interoperability with C	75
6.1.1	Intrinsic Types	75
6.1.2	Derived Types and struct	75
6.1.3	Interoperable Global Variables	76
6.1.4	Interoperable Subroutines and Functions	76
6.1.5	Working with C Pointers	78
6.1.6	Further Interoperability of Fortran with C	80
6.1.7	Generating C prototypes from Fortran	80
6.2	GNU Fortran Compiler Directives	80
6.2.1	ATTRIBUTES directive	81
6.2.2	UNROLL directive	82
6.2.3	BUILTIN directive	82
6.2.4	IVDEP directive	82
6.2.5	VECTOR directive	83
6.2.6	NOVECTOR directive	83
6.3	Non-Fortran Main Program	83
6.3.1	_gfortran_set_args — Save command-line arguments ...	84
6.3.2	_gfortran_set_options — Set library option flags	84
6.3.3	_gfortran_set_convert — Set endian conversion	86
6.3.4	_gfortran_set_record_marker — Set length of record markers	86
6.3.5	_gfortran_set_fpe — Enable floating point exception traps ..	86
6.3.6	_gfortran_set_max_subrecord_ length — Set subrecord length	87
6.4	Naming and argument-passing conventions	87
6.4.1	Naming conventions	87
6.4.2	Argument passing conventions	88

7	Coarray Programming	91
7.1	Type and enum ABI Documentation	91
7.1.1	caf_token_t	91
7.1.2	caf_register_t	91
7.1.3	caf_deregister_t	91
7.1.4	caf_reference_t	91
7.1.5	caf_team_t	93
7.2	Function ABI Documentation	93
7.2.1	_gfortran_caf_init — Initialization function	93
7.2.2	_gfortran_caf_finish — Finalization function	94
7.2.3	_gfortran_caf_this_image — Querying the image number	94
7.2.4	_gfortran_caf_num_images — Querying the maximal number of images	94
7.2.5	_gfortran_caf_image_status — Query the status of an image	95
7.2.6	_gfortran_caf_failed_images — Get an array of the indexes of the failed images	95
7.2.7	_gfortran_caf_stopped_images — Get an array of the indexes of the stopped images	96
7.2.8	_gfortran_caf_register — Registering coarrays	96
7.2.9	_gfortran_caf_deregister — Deregistering coarrays	97
7.2.10	_gfortran_caf_register_accessor — Register an accessor for remote access	98
7.2.11	_gfortran_caf_register_accessors_finish — Finish registering accessor functions	98
7.2.12	_gfortran_caf_get_remote_function_ index — Get the index of an accessor	99
7.2.13	_gfortran_caf_get_from_remote — Getting data from a remote image using a remote side accessor	99
7.2.14	_gfortran_caf_is_present_on_remote — Check that a coarray or a part of it is allocated on the remote image	101
7.2.15	_gfortran_caf_send_to_remote — Send data to a remote image using a remote side accessor to store it	102
7.2.16	_gfortran_caf_transfer_between_remotes — Initiate data transfer between to remote images	103
7.2.17	_gfortran_caf_sendget_by_ref — Sending data between remote images using enhanced references on both sides	106
7.2.18	_gfortran_caf_lock — Locking a lock variable	107
7.2.19	_gfortran_caf_lock — Unlocking a lock variable	108
7.2.20	_gfortran_caf_event_post — Post an event	108
7.2.21	_gfortran_caf_event_wait — Wait that an event occurred	109
7.2.22	_gfortran_caf_event_query — Query event count	109
7.2.23	_gfortran_caf_sync_all — All-image barrier	110
7.2.24	_gfortran_caf_sync_images — Barrier for selected images	110

8.287	TINY — Smallest positive number of a real kind	306
8.288	TRAILZ — Number of trailing zero bits of an integer	306
8.289	TRANSFER — Transfer bit patterns	307
8.290	TRANSPOSE — Transpose an array of rank two	308
8.291	TRIM — Remove trailing blank characters of a string	308
8.292	TTYNAM — Get the name of a terminal device	309
8.293	UBOUND — Upper dimension bounds of an array	309
8.294	UCOBOUND — Upper codimension bounds of an array	310
8.295	UINT — Convert to UNSIGNED type	310
8.296	UMASK — Set the file creation mask	310
8.297	UMASKL — Unsigned left justified mask	311
8.298	UMASKR — Unsigned right justified mask	311
8.299	UNLINK — Remove a file from the file system	312
8.300	UNPACK — Unpack an array of rank one into an array	312
8.301	VERIFY — Scan a string for characters not a given set	313
8.302	XOR — Bitwise logical exclusive OR	314
9	Intrinsic Modules	315
9.1	ISO_FORTRAN_ENV	315
9.2	ISO_C_BINDING	317
9.3	IEEE modules: IEEE_EXCEPTIONS, IEEE_ARITHMETIC, and IEEE_FEATURES	319
9.4	OpenMP Modules OMP_LIB and OMP_LIB_KINDS	319
9.5	OpenACC Module OPENACC	323
	Contributing	325
	Contributors to GNU Fortran	325
	Projects	326
	GNU General Public License	327
	GNU Free Documentation License	339
	ADDENDUM: How to use this License for your documents	346
	Funding Free Software	347
	Option Index	349
	Keyword Index	351

1 Introduction

This manual documents the use of `gfortran`, the GNU Fortran compiler. You can find in this manual how to invoke `gfortran`, as well as its features and incompatibilities.

Warning: This document, and the compiler it describes, are still under development. While efforts are made to keep it up-to-date, it might not accurately reflect the status of the most recent GNU Fortran compiler.

1.1 About GNU Fortran

The GNU Fortran compiler is the successor to `g77`, the Fortran 77 front end included in GCC prior to version 4 (released in 2005). While it is backward-compatible with most `g77` extensions and command-line options, `gfortran` is a completely new implementation designed to support more modern dialects of Fortran. GNU Fortran implements the Fortran 77, 90 and 95 standards completely, most of the Fortran 2003 and 2008 standards, and some features from the 2018 standard. It also implements several extensions including OpenMP and OpenACC support for parallel programming.

The GNU Fortran compiler passes the NIST Fortran 77 Test Suite (http://www.fortran-2000.com/ArnaudRecipes/fcvs21_f95.html), and produces acceptable results on the LAPACK Test Suite (<https://www.netlib.org/lapack/faq.html>). It also provides respectable performance on the Polyhedron Fortran compiler benchmarks (https://polyhedron.com/?page_id=175) and the Livermore Fortran Kernels test (<https://www.netlib.org/benchmark/livermore>). It has been used to compile a number of large real-world programs, including the HARMONIE and HIRLAM weather forecasting code (<http://hirlam.org/>) and the Tonto quantum chemistry package (<https://github.com/dylan-jayatilaka/tonto>); see <https://gcc.gnu.org/wiki/GfortranApps> for an extended list.

GNU Fortran provides the following functionality:

- Read a program, stored in a file and containing *source code* instructions written in Fortran 77.
- Translate the program into instructions a computer can carry out more quickly than it takes to translate the original Fortran instructions. The result after compilation of a program is *machine code*, which is efficiently translated and processed by a machine such as your computer. Humans usually are not as good writing machine code as they are at writing Fortran (or C++, Ada, or Java), because it is easy to make tiny mistakes writing machine code.
- Provide information about the reasons why the compiler may be unable to create a binary from the source code, for example if the source code is flawed. The Fortran language standards require that the compiler can point out mistakes in your code. An incorrect usage of the language causes an *error message*.

The compiler also attempts to diagnose cases where your program contains a correct usage of the language, but instructs the computer to do something questionable. This kind of diagnostic message is called a *warning message*.

- Provide optional information about the translation passes from the source code to machine code. This can help you to find the cause of certain bugs which may not be

obvious in the source code, but may be more easily found at a lower level compiler output. It also helps developers to find bugs in the compiler itself.

- Provide information in the generated machine code that can make it easier to find bugs in the program (using a debugging tool, called a *debugger*, such as the GNU Debugger *gdb*).
- Locate and gather machine code already generated to perform actions requested by statements in the program. This machine code is organized into *modules* and is located and *linked* to the user program.

The GNU Fortran compiler consists of several components:

- A version of the `gcc` command (which also might be installed as the system's `cc` command) that also understands and accepts Fortran source code. The `gcc` command is the *driver* program for all the languages in the GNU Compiler Collection (GCC); With `gcc`, you can compile the source code of any language for which a front end is available in GCC.
- The `gfortran` command itself, which also might be installed as the system's `f95` command. `gfortran` is just another driver program, but specifically for the Fortran compiler only. The primary difference between the `gcc` and `gfortran` commands is that the latter automatically links the correct libraries to your program.
- A collection of run-time libraries. These libraries contain the machine code needed to support capabilities of the Fortran language that are not directly provided by the machine code generated by the `gfortran` compilation phase, such as intrinsic functions and subroutines, and routines for interaction with files and the operating system.
- The Fortran compiler itself, (`f951`). This is the GNU Fortran parser and code generator, linked to and interfaced with the GCC backend library. `f951` “translates” the source code to assembler code. You would typically not use this program directly; instead, the `gcc` or `gfortran` driver programs call it for you.

1.2 GNU Fortran and GCC

GNU Fortran is a part of GCC, the *GNU Compiler Collection*. GCC consists of a collection of front ends for various languages, which translate the source code into a language-independent form called *GENERIC*. This is then processed by a common middle end which provides optimization, and then passed to one of a collection of back ends which generate code for different computer architectures and operating systems.

Functionally, this is implemented with a driver program (`gcc`) which provides the command-line interface for the compiler. It calls the relevant compiler front-end program (e.g., `f951` for Fortran) for each file in the source code, and then calls the assembler and linker as appropriate to produce the compiled output. In a copy of GCC that has been compiled with Fortran language support enabled, `gcc` recognizes files with `.f`, `.for`, `.ftn`, `.f90`, `.f95`, `.f03` and `.f08` extensions as Fortran source code, and compiles it accordingly. A `gfortran` driver program is also provided, which is identical to `gcc` except that it automatically links the Fortran runtime libraries into the compiled program.

Source files with `.f`, `.for`, `.fpp`, `.ftn`, `.F`, `.FOR`, `.FPP`, and `.FTN` extensions are treated as fixed form. Source files with `.f90`, `.f95`, `.f03`, `.f08`, `.F90`, `.F95`, `.F03` and `.F08` extensions are treated as free form. The capitalized versions of either form are run through

preprocessing. Source files with the lower case `.fpp` extension are also run through preprocessing.

This manual specifically documents the Fortran front end, which handles the programming language's syntax and semantics. The aspects of GCC that relate to the optimization passes and the back-end code generation are documented in the GCC manual; see Section "Introduction" in *Using the GNU Compiler Collection (GCC)*. The two manuals together provide a complete reference for the GNU Fortran compiler.

1.3 Standards

Fortran is developed by the Working Group 5 of Sub-Committee 22 of the Joint Technical Committee 1 of the International Organization for Standardization and the International Electrotechnical Commission (IEC). This group is known as WG5 (<http://www.nag.co.uk/sc22wg5/>). Official Fortran standard documents are available for purchase from ISO; a collection of free documents (typically final drafts) are also available on the wiki (<https://gcc.gnu.org/wiki/GFortranStandards>).

The GNU Fortran compiler implements ISO/IEC 1539:1997 (Fortran 95). As such, it can also compile essentially all standard-compliant Fortran 90 and Fortran 77 programs. It also supports the ISO/IEC TR-15581 enhancements to allocatable arrays.

GNU Fortran also supports almost all of ISO/IEC 1539-1:2004 (Fortran 2003) and ISO/IEC 1539-1:2010 (Fortran 2008). It has partial support for features introduced in ISO/IEC 1539:2018 (Fortran 2018), the most recent version of the Fortran language standard, including full support for the Technical Specification **Further Interoperability of Fortran with C** (ISO/IEC TS 29113:2012). More details on support for these standards can be found in the following sections of the documentation.

1.3.1 Fortran 95 status

The Fortran 95 standard specifies in Part 2 (ISO/IEC 1539-2:2000) varying length character strings. While GNU Fortran currently does not support such strings directly, there exist two Fortran implementations for them, which work with GNU Fortran. One can be found at <http://user.astro.wisc.edu/~townsend/static.php?ref=iso-varying-string>.

Deferred-length character strings of Fortran 2003 supports part of the features of `ISO_VARYING_STRING` and should be considered as replacement. (Namely, allocatable or pointers of the type `character(len=:)`.)

Part 3 of the Fortran 95 standard (ISO/IEC 1539-3:1998) defines Conditional Compilation, which is not widely used and not directly supported by the GNU Fortran compiler. You can use the program `coco` to preprocess such files (<http://www.daniellnagle.com/coco.html>).

1.3.2 Fortran 2003 status

GNU Fortran implements the Fortran 2003 (ISO/IEC 1539-1:2004) standard except for finalization support, which is incomplete. See the wiki page (<https://gcc.gnu.org/wiki/Fortran2003>) for a full list of new features introduced by Fortran 2003 and their implementation status.

1.3.3 Fortran 2008 status

The GNU Fortran compiler supports almost all features of Fortran 2008; the wiki (<https://gcc.gnu.org/wiki/Fortran2008Status>) has some information about the current implementation status. In particular, the following are not yet supported:

- `DO CONCURRENT` and `FORALL` do not recognize a type-spec in the loop header.
- The change to permit any constant expression in subscripts and nested implied-do limits in a `DATA` statement has not been implemented.

1.3.4 Fortran 2018 status

Fortran 2018 (ISO/IEC 1539:2018) is the most recent version of the Fortran language standard. GNU Fortran implements some of the new features of this standard:

- All Fortran 2018 features derived from ISO/IEC TS 29113:2012, “Further Interoperability of Fortran with C”, are supported by GNU Fortran. This includes assumed-type and assumed-rank objects and the `SELECT RANK` construct as well as the parts relating to `BIND(C)` functions. See also Section 6.1.6 [Further Interoperability of Fortran with C], page 80.
- GNU Fortran supports a subset of features derived from ISO/IEC TS 18508:2015, “Additional Parallel Features in Fortran”:
 - The new atomic `ADD`, `CAS`, `FETCH` and `ADD/OR/XOR`, `OR` and `XOR` intrinsics.
 - The `CO_MIN` and `CO_MAX` and `SUM` reduction intrinsics, and the `CO_BROADCAST` and `CO_REDUCE` intrinsic, except that those do not support polymorphic types or types with allocatable, pointer or polymorphic components.
 - Events (`EVENT POST`, `EVENT WAIT`, `EVENT_QUERY`).
 - Failed images (`FAIL IMAGE`, `IMAGE_STATUS`, `FAILED_IMAGES`, `STOPPED_IMAGES`).
- An `ERROR STOP` statement is permitted in a `PURE` procedure.
- GNU Fortran supports the `IMPLICIT NONE` statement with an `implicit-none-spec-list`.
- The behavior of the `INQUIRE` statement with the `RECL=` specifier now conforms to Fortran 2018.

Part I: Invoking GNU Fortran

2 GNU Fortran Command Options

The `gfortran` command supports all the options supported by the `gcc` command. Only options specific to GNU Fortran are documented here.

See Section “GCC Command Options” in *Using the GNU Compiler Collection (GCC)*, for information on the non-Fortran-specific aspects of the `gcc` command (and, therefore, the `gfortran` command).

All GCC and GNU Fortran options are accepted both by `gfortran` and by `gcc` (as well as any other drivers built at the same time, such as `g++`), since adding GNU Fortran to the GCC distribution enables acceptance of GNU Fortran options by all of the relevant drivers.

In some cases, options have positive and negative forms; the negative form of `-ffoo` would be `-fno-foo`. This manual documents only one of these two forms, whichever one is not the default.

2.1 Option summary

Here is a summary of all the options specific to GNU Fortran, grouped by type. Explanations are in the following sections.

Fortran Language Options

See Section 2.2 [Options controlling Fortran dialect], page 9.

```
-fall-intrinsics -fallow-argument-mismatch -fallow-invalid-boz
-fbackslash -fcray-pointer -fd-lines-as-code -fd-lines-as-comments
-fdec -fdec-char-conversions -fdec-structure -fdec-intrinsic-ints
-fdec-static -fdec-math -fdec-include -fdec-format-defaults
-fdec-blank-format-item -fdefault-double-8 -fdefault-integer-8
-fdefault-real-8 -fdefault-real-10 -fdefault-real-16 -fdollar-ok
-ffixed-line-length-n -ffixed-line-length-none -fpad-source
-ffree-form -ffree-line-length-n -ffree-line-length-none
-fimplicit-none -finteger-4-integer-8 -fmax-identifier-length
-fmodule-private -ffixed-form -fno-range-check -fopenacc -fopenmp
-fopenmp-allocators -fopenmp-simd -freal-4-real-10 -freal-4-real-16
-freal-4-real-8 -freal-8-real-10 -freal-8-real-16 -freal-8-real-4
-std=std -ftest-forall-temp -funsigned
```

Preprocessing Options

See Section 2.3 [Enable and customize preprocessing], page 15.

```
-A-question[=answer]
-Aquestion=answer -C -CC -Dmacro[=defn]
-H -P
-Umacro -cpp -dD -dI -dM -dN -dU -fworking-directory
-imultilib dir
-iprefix file -iquote -isysroot dir -isystem dir -nocpp
-nostdinc
-undef
```

Error and Warning Options

See Section 2.4 [Options to request or suppress errors and warnings], page 18.

```
-Waliasing -Wall -Wampersand -Warray-bounds
-Wc-binding-type -Wcharacter-truncation -Wconversion
-Wno-deprecated-openmp -Wdo-subscript -Wfunction-elimination
```


`-fdump-fortran-original -fdump-parse-tree -save-temps`

2.2 Options controlling Fortran dialect

The following options control the details of the Fortran dialect accepted by the compiler:

`-ffree-form`

`-ffixed-form`

Specify the layout used by the source file. The free form layout was introduced in Fortran 90. Fixed form was traditionally used in older Fortran programs. When neither option is specified, the source form is determined by the file extension.

`-fall-intrinsics`

This option causes all intrinsic procedures (including the GNU-specific extensions) to be accepted. This can be useful with `-std=` to force standard compliance but get access to the full range of intrinsics available with `gfortran`. As a consequence, `-Wintrinsics-std` is ignored and no user-defined procedure with the same name as any intrinsic is called except when it is explicitly declared `EXTERNAL`.

`-fallow-argument-mismatch`

Some code contains calls to external procedures with mismatches between the calls and the procedure definition, or with mismatches between different calls. Such code is nonconforming, and is usually flagged with an error. This options degrades the error to a warning that can only be disabled by disabling all warnings via `-w`. Only a single occurrence per argument is flagged by this warning. `-fallow-argument-mismatch` is implied by `-std=legacy`.

Using this option is *strongly* discouraged. It is possible to provide standard-conforming code that allows different types of arguments by using an explicit interface and `TYPE(*)`.

`-fallow-invalid-boz`

A BOZ literal constant can occur in a limited number of contexts in standard conforming Fortran. This option degrades an error condition to a warning, and allows a BOZ literal constant to appear where the Fortran standard would otherwise prohibit its use.

`-fd-lines-as-code`

`-fd-lines-as-comments`

Enable special treatment for lines beginning with `d` or `D` in fixed form sources. If the `-fd-lines-as-code` option is given they are treated as if the first column contained a blank. If the `-fd-lines-as-comments` option is given, they are treated as comment lines.

`-fdec`

DEC compatibility mode. Enables extensions and other features that mimic the default behavior of older compilers (such as DEC). These features are non-standard and should be avoided at all costs. For details on GNU Fortran's implementation of these extensions see the full documentation.

Application Programming Interface v2.6 <https://www.openacc.org>. This option implies `-pthread`, and thus is only supported on targets that have support for `-pthread`. The option `-fopenacc` implies `-frecursive`.

-fopenmp Enable handling of OpenMP directives ‘`!$omp`’ in Fortran. It additionally enables the conditional compilation sentinel ‘`!$`’ in Fortran. In fixed source form Fortran, the sentinels can also start with ‘`c`’ or ‘`*`’. When `-fopenmp` is specified, the compiler generates parallel code according to the OpenMP Application Program Interface v4.5 <https://www.openmp.org>. This option implies `-pthread`, and thus is only supported on targets that have support for `-pthread`. `-fopenmp` implies `-fopenmp-simd` and `-frecursive`.

-fopenmp-allocators

Enables handling of allocation, reallocation and deallocation of Fortran allocatable and pointer variables that are allocated using the ‘`!$omp allocators`’ and ‘`!$omp allocate`’ constructs. Files containing either directive have to be compiled with this option in addition to `-fopenmp`. Additionally, all files that might deallocate or reallocate a variable that has been allocated with an OpenMP allocator have to be compiled with this option. This includes intrinsic assignment to allocatable variables when reallocation may occur and deallocation due to either of the following: end of scope, explicit deallocation, ‘`intent(out)`’, deallocation of allocatable components etc. Files not changing the allocation status or only for components of a derived type that have not been allocated using those two directives do not need to be compiled with this option. Nor do files that handle such variables after they have been deallocated or allocated by the normal Fortran allocator.

-fopenmp-simd

Enable handling of OpenMP’s `simd`, `declare simd`, `declare reduction`, `assume`, `ordered`, `scan` and `loop` directive, and of combined or composite directives with `simd` as constituent with `!$omp` in Fortran. It additionally enables the conditional compilation sentinel ‘`!$`’ in Fortran. In fixed source form Fortran, the sentinels can also start with ‘`c`’ or ‘`*`’. Other OpenMP directives are ignored. Unless `-fopenmp` is additionally specified, the `loop` region binds to the current task region, independent of the specified `bind` clause.

-fno-range-check

Disable range checking on results of simplification of constant expressions during compilation. For example, GNU Fortran gives an error at compile time when simplifying `a = 1. / 0.` With this option, no error is given and `a` is assigned the value `+Infinity`. If an expression evaluates to a value outside of the relevant range of `[-HUGE():HUGE()]`, then the expression is replaced by `-Inf` or `+Inf` as appropriate. Similarly, `DATA i/Z'FFFFFFFF'/` results in an integer overflow on most systems, but with `-fno-range-check` the value “wraps around” and `i` is initialized to `-1` instead.

-fdefault-integer-8

Set the default integer and logical types to an 8 byte wide type. This option also affects the kind of integer constants like 42. Unlike **-finteger-4-integer-8**, it does not promote variables with explicit kind declaration.

-fdefault-real-8

Set the default real type to an 8 byte wide type. This option also affects the kind of non-double real constants like 1.0. This option promotes the default width of DOUBLE PRECISION and double real constants like 1.d0 to 16 bytes if possible. If **-fdefault-double-8** is given along with **fdefault-real-8**, DOUBLE PRECISION and double real constants are not promoted. Unlike **-freal-4-real-8**, **fdefault-real-8** does not promote variables with explicit kind declarations.

-fdefault-real-10

Set the default real type to an 10 byte wide type. This option also affects the kind of non-double real constants like 1.0. This option promotes the default width of DOUBLE PRECISION and double real constants like 1.d0 to 16 bytes if possible. If **-fdefault-double-8** is given along with **fdefault-real-10**, DOUBLE PRECISION and double real constants are not promoted. Unlike **-freal-4-real-10**, **fdefault-real-10** does not promote variables with explicit kind declarations.

-fdefault-real-16

Set the default real type to an 16 byte wide type. This option also affects the kind of non-double real constants like 1.0. This option promotes the default width of DOUBLE PRECISION and double real constants like 1.d0 to 16 bytes if possible. If **-fdefault-double-8** is given along with **fdefault-real-16**, DOUBLE PRECISION and double real constants are not promoted. Unlike **-freal-4-real-16**, **fdefault-real-16** does not promote variables with explicit kind declarations.

-fdefault-double-8

Set the DOUBLE PRECISION type and double real constants like 1.d0 to an 8 byte wide type. Do nothing if this is already the default. This option prevents **-fdefault-real-8**, **-fdefault-real-10**, and **-fdefault-real-16**, from promoting DOUBLE PRECISION and double real constants like 1.d0 to 16 bytes.

-finteger-4-integer-8

Promote all INTEGER(KIND=4) entities to an INTEGER(KIND=8) entities. If KIND=8 is unavailable, then an error is issued. This option should be used with care and may not be suitable for your codes. Areas of possible concern include calls to external procedures, alignment in EQUIVALENCE and/or COMMON, generic interfaces, BOZ literal constant conversion, and I/O. Inspection of the intermediate representation of the translated Fortran code, produced by **-fdump-tree-original**, is suggested.

`-freal-4-real-8`
`-freal-4-real-10`
`-freal-4-real-16`
`-freal-8-real-4`
`-freal-8-real-10`
`-freal-8-real-16`

Promote all `REAL(KIND=M)` entities to `REAL(KIND=N)` entities. If `REAL(KIND=N)` is unavailable, then an error is issued. The `-freal-4-` flags also affect the default real kind and the `-freal-8-` flags also the double-precision real kind. All other real-kind types are unaffected by this option. The promotion is also applied to real literal constants of default and double-precision kind and a specified kind number of 4 or 8, respectively. However, `-fdefault-real-8`, `-fdefault-real-10`, `-fdefault-real-16`, and `-fdefault-double-8` take precedence for the default and double-precision real kinds, both for real literal constants and for declarations without a kind number. Note that for `REAL(KIND=KIND(1.0))` the literal may get promoted and then the result may get promoted again. These options should be used with care and may not be suitable for your codes. Areas of possible concern include calls to external procedures, alignment in `EQUIVALENCE` and/or `COMMON`, generic interfaces, `BOZ` literal constant conversion, and I/O and calls to intrinsic procedures when passing a value to the `kind=` dummy argument. Inspection of the intermediate representation of the translated Fortran code, produced by `-fdump-fortran-original` or `-fdump-tree-original`, is suggested. NOTE: These options are incompatible with user defined derived type I/O (DTIO).

`-std=std` Specify the standard to which the program is expected to conform, which may be one of ‘f95’, ‘f2003’, ‘f2008’, ‘f2018’, ‘f2023’, ‘gnu’, or ‘legacy’. The default value for *std* is ‘gnu’, which specifies a superset of the latest Fortran standard that includes all of the extensions supported by GNU Fortran, although warnings are given for obsolete extensions not recommended for use in new code. The ‘legacy’ value is equivalent but without the warnings for obsolete extensions, and may be useful for old nonstandard programs. The ‘f95’, ‘f2003’, ‘f2008’, ‘f2018’, and ‘f2023’ values specify strict conformance to the Fortran 95, Fortran 2003, Fortran 2008, Fortran 2018 and Fortran 2023 standards, respectively; errors are given for all extensions beyond the relevant language standard, and warnings are given for the Fortran 77 features that are permitted but obsolescent in later standards. The deprecated option ‘`-std=f2008ts`’ acts as an alias for ‘`-std=f2018`’. It is only present for backwards compatibility with earlier gfortran versions and should not be used any more. ‘`-std=f202y`’ acts as an alias for ‘`-std=f2023`’ and enables proposed features for testing Fortran 202y. As the Fortran 202y standard develops, implementation might change or the experimental new features might be removed.

`-ftest-forall-temp`

Enhance test coverage by forcing most forall assignments to use temporary.

`-funsigned`

Allow the experimental unsigned extension.

- dD Like -dM except in two respects: it does not include the predefined macros, and it outputs both the `#define` directives and the result of preprocessing. Both kinds of output go to the standard output file.
- dN Like -dD, but emit only the macro names, not their expansions.
- dU Like dD except that only macros that are expanded, or whose definedness is tested in preprocessor directives, are output; the output is delayed until the use or test of the macro; and `#undef` directives are also output for macros tested but undefined at the time.
- dI Output `#include` directives in addition to the result of preprocessing.
- fworking-directory
 Enable generation of linemarkers in the preprocessor output that let the compiler know the current working directory at the time of preprocessing. When this option is enabled, the preprocessor emits, after the initial linemarker, a second linemarker with the current working directory followed by two slashes. GCC uses this directory, when it is present in the preprocessed input, as the directory emitted as the current working directory in some debugging information formats. This option is implicitly enabled if debugging information is enabled, but this can be inhibited with the negated form `-fno-working-directory`. If the `-P` flag is present in the command line, this option has no effect, since no `#line` directives are emitted whatsoever.
- idirafter *dir*
 Search *dir* for include files, but do it after all directories specified with `-I` and the standard system directories have been exhausted. *dir* is treated as a system include directory. If *dir* begins with `=`, then the `=` is replaced by the sysroot prefix; see `--sysroot` and `-isysroot`.
- imultilib *dir*
 Use *dir* as a subdirectory of the directory containing target-specific C++ headers.
- iprefix *prefix*
 Specify *prefix* as the prefix for subsequent `-iwithprefix` options. If the *prefix* represents a directory, you should include the final `'/'`.
- isysroot *dir*
 This option is like the `--sysroot` option, but applies only to header files. See the `--sysroot` option for more information.
- iquote *dir*
 Search *dir* only for header files requested with `#include "file"`; they are not searched for `#include <file>`, before all directories specified by `-I` and before the standard system directories. If *dir* begins with `=`, then the `=` is replaced by the sysroot prefix; see `--sysroot` and `-isysroot`.
- isystem *dir*
 Search *dir* for header files, after all directories specified by `-I` but before the standard system directories. Mark it as a system directory, so that it gets the same special treatment as is applied to the standard system directories. If *dir*

begins with `=`, then the `=` is replaced by the `sysroot` prefix; see `--sysroot` and `-isysroot`.

-nostdinc

Do not search the standard system directories for header files. Only the directories you have specified with `-I` options (and the directory of the current file, if appropriate) are searched.

-undef Do not predefine any system-specific or GCC-specific macros. The standard predefined macros remain defined.

-Apredicate=answer

Make an assertion with the predicate *predicate* and answer *answer*. This form is preferred to the older form `-A predicate(answer)`, which is still supported, because it does not use shell special characters.

-A-predicate=answer

Cancel an assertion with the predicate *predicate* and answer *answer*.

-C Do not discard comments. All comments are passed through to the output file, except for comments in processed directives, which are deleted along with the directive.

You should be prepared for side effects when using `-C`; it causes the preprocessor to treat comments as tokens in their own right. For example, comments appearing at the start of what would be a directive line have the effect of turning that line into an ordinary source line, since the first token on the line is no longer a `#`.

Warning: this currently handles C-Style comments only. The preprocessor does not yet recognize Fortran-style comments.

-CC Do not discard comments, including during macro expansion. This is like `-C`, except that comments contained within macros are also passed through to the output file where the macro is expanded.

In addition to the side-effects of the `-C` option, the `-CC` option causes all C++-style comments inside a macro to be converted to C-style comments. This is to prevent later use of that macro from inadvertently commenting out the remainder of the source line. The `-CC` option is generally used to support lint comments.

Warning: this currently handles C- and C++-Style comments only. The preprocessor does not yet recognize Fortran-style comments.

-Dname Predefine name as a macro, with definition 1.

-Dname=definition

The contents of *definition* are tokenized and processed as if they appeared during translation phase three in a `#define` directive. In particular, the definition is truncated by embedded newline characters.

If you are invoking the preprocessor from a shell or shell-like program you may need to use the shell's quoting syntax to protect characters such as spaces that have a meaning in the shell syntax.

If you wish to define a function-like macro on the command line, write its argument list with surrounding parentheses before the equals sign (if any). Parentheses are meaningful to most shells, so you need to quote the option. With sh and csh, `-D'name(args...)=definition'` works.

`-D` and `-U` options are processed in the order they are given on the command line. All `-imacros` file and `-include` file options are processed after all `-D` and `-U` options.

- `-H` Print the name of each header file used, in addition to other normal activities. Each name is indented to show how deep in the `'#include'` stack it is.
- `-P` Inhibit generation of linemarkers in the output from the preprocessor. This might be useful when running the preprocessor on something that is not C code, and is sent to a program that might be confused by the linemarkers.
- `-Uname` Cancel any previous definition of *name*, either built in or provided with a `-D` option.

2.4 Options to request or suppress errors and warnings

Errors are diagnostic messages that report that the GNU Fortran compiler cannot compile the relevant piece of source code. The compiler continues to process the program in an attempt to report further errors to aid in debugging, but does not produce any compiled output.

Warnings are diagnostic messages that report constructions that are not inherently erroneous but that are risky or suggest there is likely to be a bug in the program. Unless `-Werror` is specified, they do not prevent compilation of the program.

You can request many specific warnings with options beginning `-W`, for example `-Wimplicit` to request warnings on implicit declarations. Each of these specific warning options also has a negative form beginning `-Wno-` to turn off warnings; for example, `-Wno-implicit`. This manual lists only one of the two forms, whichever is not the default.

These options control the amount and kinds of errors and warnings produced by GNU Fortran:

`-fmax-errors=n`

Limits the maximum number of error messages to *n*, at which point GNU Fortran bails out rather than attempting to continue processing the source code. If *n* is 0, there is no limit on the number of error messages produced.

`-fsyntax-only`

Check the code for syntax errors, but do not actually compile it. This generates module files for each module present in the code, but no other output file.

`-Wpedantic`

`-pedantic`

Issue warnings for uses of extensions to Fortran. `-pedantic` also applies to C-language constructs where they occur in GNU Fortran source files, such as use of `'\e'` in a character constant within a directive like `#include`.

-Wno-overwrite-recursive

Do not warn when `-fno-automatic` is used with `-frecursive`. Recursion is broken if the relevant local variables do not have the attribute `AUTOMATIC` explicitly declared. This option can be used to suppress the warning when it is known that recursion is not broken. Useful for build environments that use `-Werror`.

-Wreal-q-constant

Produce a warning if a real-literal-constant contains a `q` exponent-letter.

-Wsurprising

Produce a warning when “suspicious” code constructs are encountered. While technically legal these usually indicate that an error has been made.

This currently produces a warning under the following circumstances:

- An `INTEGER`-typed `SELECT CASE` construct has a `CASE` that can never be matched as its lower value is greater than its upper value.
- A `LOGICAL`-typed `SELECT CASE` construct has three `CASE` statements.
- A `TRANSFER` specifies a source that is shorter than the destination.
- The type of a function result is declared more than once with the same type. If `-pedantic` or standard-conforming mode is enabled, this is an error.
- A `CHARACTER` variable is declared with negative length.
- With `-fopenmp`, for fixed-form source code, when an `omx` vendor-extension sentinel is encountered. (The equivalent `ompx`, used in free-form source code, is diagnosed by default.)
- With `-fopenacc`, when using named constances with clauses that take a variable as doing so has no effect.

-Wtabs

By default, tabs are accepted as whitespace, but tabs are not members of the Fortran Character Set. For continuation lines, a tab followed by a digit between 1 and 9 is supported. `-Wtabs` causes a warning to be issued if a tab is encountered. Note, `-Wtabs` is active for `-pedantic`, `-std=f95`, `-std=f2003`, `-std=f2008`, `-std=f2018`, `-std=f2023` and `-Wall`.

-Wundefined-do-loop

Warn if a `DO` loop with step either 1 or -1 yields an underflow or an overflow during iteration of an induction variable of the loop. This option is implied by `-Wall`.

-Wundefined-vars

Warn if variables are found to be used that cannot be defined (have a value assigned to them). This also includes using allocatable variables that are not allocated.

-Wunderflow

Produce a warning when numerical constant expressions that yield an underflow are encountered during compilation. Enabled by default.

-Wintrinsic-shadow

Warn if a user-defined procedure or module procedure has the same name as an intrinsic; in this case, an explicit interface or `EXTERNAL` or `INTRINSIC` declaration might be needed to get calls later resolved to the desired intrinsic/procedure. This option is implied by `-Wall`.

-Wuse-without-only

Warn if a `USE` statement has no `ONLY` qualifier and thus implicitly imports all public entities of the used module.

-Wunused-dummy-argument

Warn about unused dummy arguments. This option is implied by `-Wall`.

-Wunused-parameter

Contrary to `gcc`'s meaning of `-Wunused-parameter`, `gfortran`'s implementation of this option does not warn about unused dummy arguments (see `-Wunused-dummy-argument`), but about unused `PARAMETER` values. `-Wunused-parameter` is implied by `-Wextra` if also `-Wunused` or `-Wall` is used.

-Wunused-intent-out

Warn about variables passed to `INTENT(OUT)` arguments whose values are then unused. This option is implied by `-Wextra`. The following code shows an example where this warning will be emitted:

```
module y
  implicit none
  contains
    subroutine bar
      real :: a
      call foo(a) ! Warning will be emitted here
    end subroutine bar
    subroutine foo(x)
      real, intent(out) :: x
      x = 0.4
    end subroutine foo
end module y
```

-Wunused-read

Warn about variables in `READ` statements whose values are never used. This warning is implied by `-Wextra`. The following code shows an example where this warning will be emitted:

```
program main
  real :: x
  read (*,*) x
end program main
```

-Walign-commons

By default, `gfortran` warns about any occasion of variables being padded for proper alignment inside a `COMMON` block. This warning can be turned off via `-Wno-align-commons`. See also `-falign-commons`.

-Wfunction-elimination

Warn if any calls to impure functions are eliminated by the optimizations enabled by the `-ffrontend-optimize` option. This option is implied by `-Wextra`.

termination. When the environment variable is not set, then the number of hardware threads reported by the OS is used. Over-provisioning is possible. The number of images is limited only by the OS and the size of an integer variable on the architecture the program is running on.

GFORTTRAN_SHARED_MEMORY_SIZE: The size of the shared-memory segment made available to all images is fixed and needs to be set at program start. It cannot grow or shrink. The size can be given in bytes (no suffix), kilobytes (**k** or **K** suffix), megabytes (**m** or **M**) or gigabytes (**g** or **G**). If the variable is not set, or not parseable, then on 32-bit architectures 2^{28} bytes and on 64-bit 2^{34} bytes are chosen. Note, although the size is set, most modern systems do not allocate the memory at program start. This allows one to choose a shared-memory size larger than available memory.

Warning: Choosing a large shared-memory size may produce large core dumps!

GFORTTRAN_IMAGE_RESTARTS_LIMIT: On certain platforms, esp. MacOS, the shared-memory segment needs to be placed on the same (virtual) address in every image or synchronization primitives do not work as expected. Unfortunately some operating systems are somewhat arbitrary on when they can do this. When the OS is not able to fulfill the request, the image aborts itself and is restarted by the supervisor until the OS complies. This environment variable limits the total number of restarts of all images having an issue with shared-memory segment placement. The default value is 4000.

The shared-memory coarray library internally uses some additional environment variables, which are overwritten without notice or may result in failure to start. These are: **GFORTTRAN_IMAGE_NUM**, **GFORTTRAN_SHMEM_PID**, and **GFORTTRAN_SHMEM_BASE**. Using these variables is strongly discouraged. Special care needs to be taken when one coarray program starts another coarray program as a child process. In this case it is the spawning process' responsibility to remove the above variables from the environment.

Part II: Language Reference
