NAG Fortran Compiler Release 5.3

July 31, 2012

1 Name

nagfor — NAG Fortran Compiler Release 5.3

2 Usage

nagfor [mode] [option]... *file*...

3 Description

nagfor is the interface to the NAG Fortran Compiler system. The compiler translates programs written in Fortran into executable programs, relocatable binary modules, assembler source files or C source files.

The *mode* determines the action performed, and can be one of

=compiler
: Compile (and/or link) the files; this is the default mode if none is specified.

=callgraph
: Produce a callgraph of the Fortran routines in the files (see the Producing a Call Graph section).

=depend
: Produce a dependency analysis of the Fortran files (see the Dependency Analysis section).

=polish Pretty-print (polish) the Fortran files (see the Source File Polishing section).

Options that do not apply to the current mode of operation (e.g. polish options when the mode is for compilation) are ignored.

4 File Types

A file ending in '`.f90`' or '`.f95`' is taken to be a Fortran free-form source file, a file ending in '`.f`', '`.for`' or '`.ftn`' is taken to be a Fortran fixed-form source file; these assumptions can be overridden with the *–fixed* or *–free* option. A file ending in '`.ff90`' or '`.ff95`' is taken to be a free-form file requiring preprocessing by fpp, and a file ending in '`.ff`' is taken to be a fixed-form file requiring preprocessing by fpp. A file ending in '`.F90`' or '`.F95`' is taken to be a free-form file requiring preprocessing by fpp, and a file ending in '`.F`' is taken to be a fixed-form files requiring preprocessing by fpp.

If a filename without a suffix is provided nagfor will look for a file with the suffix '`.f95`', and if that does not exist, the suffix '`.f90`'.

Non-intrinsic modules and `INCLUDE` files are expected to exist in the current working directory or in a directory named by the $-I$ option.

5 Compiler Options

–132 Increase the length of each fixed source form input line from 72 characters to 132 characters. This has no effect on free source form input.

–B*binding*

Specify **static** or **dynamic** binding. This only has effect if specified during the link phase. The default is dynamic binding. These options are positional and can be used to selectively bind some libraries statically and some dynamically.

–c Compile only (produce .o file for each source file), do not link the .o files to produce an executable file. This option is equivalent to *−otype=obj*.

–C Compile with all but the most expensive runtime checks; this omits the *−C=dangling* and *−C=undefined* options.

–C=*check*

Compile checking code according to the value of *check*, which must be one of:

all	(perform all checks except for *−C=undefined*),
array	(check array bounds),
bits	(check bit intrinsic arguments),
calls	(check procedure references),
dangling	(check for dangling pointers),
do	(check DO loops for zero step values),
none	(do no checking: this is the default),
present	(check OPTIONAL references),
pointer	(check POINTER references),
recursion	(check for invalid recursion) or
undefined	(check for undefined variables).

The *−C=undefined* option is subject to a number of limitations and is not binary-compatible with code compiled without that option; see the Undefined Variable Detection section for further details.

–colour Colour the message output from the compiler using ANSI escape sequences and the default foreground colouring scheme which is: red for error messages (including fatal errors), blue for warning messages and green for information messages.

–colour=*scheme*

Colour the message output from the compiler according to the specified *scheme*. This is a comma-separated list of colour specifications, each consisting of a message category name ("error", "warn" or "info") followed by a colon and the foreground colour name, optionally following by a plus sign and the background colour name. The colouring for unspecified categories will be the default.

Colours are: black, red, green, yellow, blue, magenta, cyan and white.

E.g. —colour=error:red+blue,warn:cyan,info:magenta+yellow
would be a rather garish colour scheme.

–compatible

Make external linkages compatible with other compilers where possible; on Windows this is Microsoft Fortran, on Mac OSX and Linux this is g77, g95 and gfortran, and on other systems this is the operating system vendor's compiler. This affects the naming convention and procedure calling convention (for example, on Windows it causes use of the "STDCALL" calling convention that is commonly used for most DLLs).

–convert=*format*

Set the default conversion mode for unformatted files to *format*. This format may be overridden by an explicit `CONVERT=` specifier in the `OPEN` statement, or by the environment variable `FORT_CONVERT`n (where n is the unit number). The value of *format* must be one of the following (not case-sensitive):

Format	Description
`BIG_ENDIAN`	synonym for `BIG_IEEE`
`BIG_IEEE_DD`	big-endian with IEEE floating-point, quad precision is double-double
`BIG_IEEE`	big-endian with IEEE floating-point, including quad precision
`BIG_NATIVE`	big-endian with native floating-point format
`LITTLE_ENDIAN`	synonym for `LITTLE_IEEE`
`LITTLE_IEEE_DD`	little-endian with IEEE floating-point, quad precision is double-double
`LITTLE_IEEE`	little-endian with IEEE floating-point, including quad precision
`LITTLE_NATIVE`	little-endian with native floating-point format
`NATIVE`	no conversion (the default)

–D*name*

Defines *name* to fpp as a preprocessor variable. This only affects files that are being preprocessed by fpp.

–dcfuns Enable recognition of non-standard double precision complex intrinsic functions. These act as specific versions of the standard generic intrinsics as follows:

Non-standard	Equivalent Standard Fortran Generic Intrinsic Function
`CDABS(A)`	`ABS(A)`
`DCMPLX(X,Y)`	`CMPLX(X,Y,KIND=KIND(0d0))`
`DCONJG(Z)`	`CONJG(Z)`
`DIMAG(Z)`	`AIMAG(Z)`
`DREAL(Z)`	`REAL(Z)` or `DBLE(Z)`

–double Double the size of default `INTEGER`, `LOGICAL`, `REAL` and `COMPLEX`. Entities specified with explicit kind numbers or byte lengths are unaffected. If quadruple precision `REAL` is available, the size of `DOUBLE PRECISION` is also doubled.

–dryrun Show but do not execute commands constructed by the compiler driver.

–dusty Allows the compilation and execution of "legacy" software by downgrading the category of common errors found in such software from "Error" to "Warning" (which may then be suppressed entirely with the *–w* option). This option disables *–C=calls*, and also enables Hollerith i/o (see the *–hollerith_io* option).

–encoding=*charset*

Specifies that the encoding system of the Fortran source files is *charset*, which must be one of **ISO_Latin_1**, **Shift_JIS** or **UTF_8**. If this option is not specified, the default encoding is UTF-8 for Fortran source files that begin with a UTF-8 Byte Order Mark, and ISO Latin-1 (if the language setting is English) or Shift-JIS (if the language setting is Japanese) for other Fortran source files.

–english Produce compiler messages in English (default).

–F Preprocess only, do not compile. Each file that is preprocessed will produce an output file of the same name with the suffix replaced by `.f`, `.f90` or `.f95` according to the suffix of the input file. This option is equivalent to *–otype=Fortran*.

–f90_sign

Use the Fortran 77/90 version of the `SIGN` intrinsic instead of the Fortran 95 one (they differ in the treatment of negative zero).

–f95 Specify that the base language is Fortran 95. This only affects extension message generation (Fortran 2003 and 2008 features will be reported as extensions).

–f2003 Specify that the base language is Fortran 2003. This only affects extension message generation (Fortran 2008 features will be reported as extensions).

–f2008 Specify that the base language is Fortran 2008. This is the default.

–fixed Interpret all Fortran source files according to fixed-form rules.

–fpp Preprocess the source files using fpp even if the suffix would normally indicate an ordinary Fortran file.

–free Interpret all Fortran source files according to free-form rules.

–g Produce information for interactive debugging by the host system debugger.

–g90 Produce debugging information for dbx90, a Fortran 90 aware front-end to the host system debugger. This produces a debug information (.g90) file for each Fortran source file. This option must be specified for both compilation and linking.

–gc Enables automatic garbage collection of the executable program. This option must be specified for both compilation and linking, and is unavailable on IBM z9 OpenEdition and Windows x64. It is incompatible with the *–thread_safe* and *–mtrace* options. For more details see the Automatic Garbage Collection section.

–gline Compile code to produce a traceback when a runtime error message is generated. Only routines compiled with this option will appear in such a traceback. This option increases both executable file size and execution time. For example:

```
Runtime Error: Invalid input for real editing
Program terminated by I/O error on unit 5 (Input_Unit,Formatted,Sequential)
main.f90, line 28: Error occurred in READ_DATA
main.f90, line 57: Called by READ_COORDS
main.f90, line 40: Called by INITIAL
main.f90, line 13: Called by $main$
```

–help Display a one-line summary of the options available for the current mode (*=compiler*, *=callgraph*, *=depend* or *=polish*).

–hollerith_io
Enable Fortran-66 compatible input/output of character data stored in numeric variables using the `A` edit descriptor. This was superseded by the `CHARACTER` datatype in Fortran 77.

–hpf Accept the extensions to Fortran specified by the High Performance Fortran Forum in HPF 1.0. These consist of the `EXTRINSIC` keyword and a large number of compiler directives. The compiler directives are checked for correctness but have no effect on compilation.

–I *pathname*
Add *pathname* to the list of directories which are to be searched for module information (`.mod`) files and `INCLUDE` files. The current working directory is always searched first, then any directories named in *−I* options, then the compiler's library directory (see the *−Qpath* option).

–indirect *file*
Read the contents of *file* as additional arguments to the compiler driver. This option may also be given by "@*file*"; note in this case there is no space between the '@' and the file name.

In an indirect file, arguments may be given on separate lines; on a single line, multiple arguments may be separated by blanks. A blank can be included in an option or file name by putting the whole option or file name in quotes (`"`); this is the only quoting mechanism. An indirect file may reference other indirect files.

–ieee=*mode*
Set the mode of IEEE arithmetic operation according to *mode*, which must be one of **full**, **nonstd** or **stop**.

full enables all IEEE arithmetic facilities including non-stop arithmetic.

nonstd Disables non-stop arithmetic, terminating execution on floating overflow, division by zero or invalid operand. If the hardware supports it, this also disables IEEE gradual underflow, producing zero instead of a denormalised number; this can improve performance on some systems.

stop enables all IEEE arithmetic facilities except for non-stop arithmetic; execution will be terminated on floating overflow, division by zero or invalid operand.

The *−ieee* option must be specified when compiling the main program unit, and its effect is global. The default mode is *−ieee=stop*. For more details see the IEEE 754 Arithmetic Support section.

–info Request output of information messages. The default is to suppress these messages.

–kind=*option*
Specify the kind numbering system to be used; *option* must be one of **byte** or **sequential**.

For *−kind=byte*, the kind numbers for `INTEGER`, `REAL` and `LOGICAL` will match the number of bytes of storage (e.g., default `REAL` is 4 and `DOUBLE PRECISION` is 8). Note that `COMPLEX` kind numbers are the same as its `REAL` components, and thus half of the total byte length in the entity.

For *−kind=sequential* (the default), the kind numbers for all datatypes are numbered sequentially from 1, increasing with precision (e.g., default `REAL` is 1 and `DOUBLE PRECISION` is 2).

This option does not affect the interpretation of byte-length specifiers (an extension to Fortran 77).

–l*x* Link with library lib*x*.a. The linker will search for this library in the directories specified by −L*dir* options followed by the normal system directories (see the ld(1) command).

–L*dir* Add *dir* to the list of directories for library files (see the ld(1) command).

–M Produce module information files (`.mod` files) only. This option is equivalent to *−otype=mod*.

–max_parameter_size=*N*
Set the maximum size of a `PARAMETER` to *N* MB (megabytes). *N* must be in the range 1 to 1048576 (1MB to 1TB); the default is 50 MB.

–maxcontin=*N*
Increase the limit on the number of continuation lines from 255 to *N*. This option will not decrease the limit below the standard number.

–mdir *dir*
Write any module information (`.mod`) files to directory *dir* instead of the current working directory.

–mismatch
Downgrade consistency checking of procedure argument lists so that mismatches produce warning messages instead of error messages. This only affects calls to a routine which is not in the current file; calls to a routine in the file being compiled must still be correct. This option disables *–C=calls*.

–mismatch_all
Further downgrade consistency checking of procedure argument lists so that calls to routines in the same file which are incorrect will produce warnings instead of error messages. This option disables *–C=calls*.

–mtrace Trace memory allocation and deallocation. This option is a synonym for *–mtrace=on*.

–mtrace=*trace_opt_list*
Trace memory allocation and deallocation according to the value of *trace_opt_list*, which must be a comma separated list of one or more of:

address (display addresses),
all (all options except for **off**),
line (display file/line info if known),
off (disable tracing output),
on (enable tracing output),
paranoia (protect memory allocator data structures against the user program),
size (display size in bytes) or
verbose (all options except for **off** and **paranoia**).

This option should be specified during both compilation and linking, and is incompatible with the *–gc* option. For more details see the Memory Tracing section.

–nan Initialise `REAL` and `COMPLEX` variables to IEEE Signalling NaN, causing a runtime crash if the values are used before being set. This affects local variables, module variables, and `INTENT(OUT)` dummy arguments only; it does not affect variables in `COMMON` or `EQUIVALENCE`.

–nihongo
Produce compiler messages in Japanese (using Shift-JIS encoding).

–no_underflow_warning
Suppress the warning message that normally appears if a floating-point underflow occurred during execution. This option is only effective if specified during the link phase.

–nocheck_modtime
Do not check for `.mod` files being out of date.

–nomod Suppress module information (`.mod`) file production. Combining this with *–M* will produce no output (other than error and warning messages) at all, equivalent to *–otype=none*.

–noqueue
If no licence for the compiler is immediately available, exit with an error instead of queueing for it.

–o *output*
Name the output file *output* instead of the default. If an executable is being produced the default is `a.out`; otherwise it is *file*.o with the *-c* option, *file*.c with the *-S* option, and *file*.f, *file*.f90 or *file*.f95 with the *-F* option, where *file* is the base part of the source file (i.e. with the suffix removed).

–O Normal optimisation, equivalent to *–O2*.

–O*N* Set the optimisation level to *N*. The optimisation levels are:

–O0 No optimisation. This is the default, and is recommended when debugging.

–O1 Minimal quick optimisation.

–O2 Normal optimisation.

–O3 Further optimisation.

–O4 Maximal optimisation.

–Oassumed
This is a synonym for *–Oassumed=contig*.

–Oassumed=*shape*
Optimises assumed-shape array dummy arguments according to the value of *shape*, which must be one of

always_contig
Optimised for contiguous actual arguments. If the actual argument is not contiguous a runtime error will occur (the compiler is not standard-conforming under this option).

contig Optimised for contiguous actual arguments; if the actual argument is not contiguous (i.e. it is an array section) a contiguous local copy is made. This may speed up array section accessing if a sufficiently large number of array element or array operations is performed (i.e. if the cost of making the local copy is less than the overhead of discontiguous array accesses), but usually makes such accesses slower. Note that this option does not affect dummy arguments with the `TARGET` attribute; these are always accessed via the dope vector.

section Optimised for low-moderate accesses to array section (discontiguous) actual arguments. This is the default.

Note that `CHARACTER` arrays are not affected by these options.

–Oblock=*N*
Specify the dimension of the blocks used for evaluating the `MATMUL` intrinsic. The default value (only for *–O1* and above) is system and datatype dependent.

–Onopropagate
Disable the optimisation of constant propagation. This is the default for *–O1* and lower.

–Opropagate
Enable the optimisation of constant propagation. This is the default for *–O2* and higher.

–Orounding
Specify that the program does not alter the default rounding mode. This enables the use of faster code for the `ANINT` intrinsic.

–Ounroll=*N*
Specify the depth to which simple loops and array operations should be unrolled. The default is no unrolling (i.e. a depth of 1) for *–O0* and *–O1*, and a depth of 2 for *–O* and higher optimisation levels. It can be advantageous to disable the Fortran compiler's loop unrolling if the C compiler normally does a very good job itself — this can be accomplished with *–Ounroll=1*.

–Ounsafe
Perform possibly unsafe optimisations that may depend on the numerical stability of the program.

–openmp
Recognise OpenMP directives and link with the OpenMP support library. For more details see the OpenMP Support section.

–otype=*filetype*
Specify the type of output file required to *filetype*, which must be one of

c	(C source file),
exe	(executable file),
fortran	(Fortran source file),
mod	(module information file),
none	(no output file),
obj	(object file).

The *–c*, *–F* and *–M* options are equivalent to *–otype=obj*, *–otype=Fortran* and *–otype=mod* respectively.

–pg Compile code to generate profiling information which is written at run-time to an implementation-dependent file (usually *gmon.out* or *mon.out*). An execution profile may then be generated using *gprof*. This option must be specified for compilation and linking and may be unavailable on some implementations.

–pic Produce position-independent code (small model), for use in a shared library. If the shared library is too big for the small model, use *–PIC*.

–PIC Produce position-independent code (large model), for use in a shared library.

–Qpath ***pathname***
Change the compiler library pathname from its default location to *pathname*. (The default location on Solaris is usually '`/opt/NAG_Fortran/lib`'.)

–r8 Double the size of default `REAL` and `COMPLEX`, and on machines for which quadruple-precision floating-point arithmetic is available, double the size of `DOUBLE PRECISION` (and the non-standard `DOUBLE COMPLEX`). `REAL` or `COMPLEX` specified with explicit `KIND` numbers or byte lengths are unaffected — but since the `KIND` intrinsic returns the correct values, `COMPLEX(KIND(0d0))` on a machine with quad-precision floating-point will correctly select quad-precision `COMPLEX`.

This has no effect on `INTEGER` sizes, and so the compiler is not standard-conforming in this mode.

Note: This option has been superseded by the *–double* option which doubles the size of all numeric data types.

–s Strip symbol table information from the executable file. This option is only effective if specified during the link phase.

–S Produce assembler (actually C source code). The resulting .c file should be compiled with the NAG Fortran compiler, not with the C compiler directly. This option is equivalent to *–otype=c*.

–save This is equivalent to inserting the `SAVE` statement in all subprograms which are not declared `RECURSIVE`, thus causing all non-automatic local variables in such subprograms to be statically allocated.

–strict95
Produce obsolescence warning messages for use of '`CHARACTER*`' syntax. This message is not produced by default since many programs contain this syntax.

–target=*machine*
Specify the machine for which code should be generated and optimised. For Sun/SPARC, *machine* may be one of

V7	SPARCstation 1 et al,
V8	SPARCstation 2 et al,
super	SuperSPARC,
ultra	UltraSPARC or
native	the current machine.

The default is to compile for SPARC V7.

Note that programs compiled for later versions of the architecture may not run, or may run much more slowly, on an earlier machine.

–tempdir ***directory***
Set the directory used for the compiler's temporary files to *directory*. The default is to use the directory named by the `TMPDIR` environment variable, or if that is not set, /tmp.

–thread_safe
Compile code for safe execution in a multi-threaded environment. This must be specified when compiling and also during the link phase. It is incompatible with *–gc*.

–time Report execution times for the various compilation phases.

–u Specify that `IMPLICIT NONE` is in effect by default, unless overridden by explicit `IMPLICIT` statements.

–unsharedrts
Bind with the unshared (static) version of the Fortran runtime system; this allows a dynamically linked executable to be run on systems where the NAG Fortran Compiler is not installed. This option is only effective if specified during the link phase.

–v Verbose. Print the name of each file as it is compiled.

−V Print version information about the compiler.

−w Suppress all warning messages. This option is a synonym for *−w=all*.

−w=*class*

Suppress the warning messages specified by *class*, which must be one of **all**, **alloctr**, **obs**, **ques**, **uda**, **uei**, **uep**, **uip**, **ulv**, **unreffed**, **unused**, **usf**, **usy**, **x77** or **x95**.

−w=all suppresses all warning messages;

−w=alloctr suppresses warning messages about the use of allocatable components, dummy arguments and functions;

−w=obs suppresses warning messages about the use of obsolescent features;

−w=ques suppresses warning messages about questionable usage;

−w=uda suppresses warning messages about unused dummy arguments;

−w=uei suppresses warning messages about unused explicit imports;

−w=uep suppresses warning messages about unused external procedures;

−w=uip suppresses warning messages about unused intrinsic procedures;

−w=ulv suppresses warning messages about unused local variables;

−w=unreffed suppresses warning messages about variables set but never referenced;

−w=unused suppresses warning messages about unused entities — this is equivalent to '*−w=uda −w=uei −w=uep −w=uip −w=ulv −w=usf −w=usy*';

−w=usf suppresses warning messages about unused statement functions;

−w=usy suppresses warning messages about unused symbols;

−w=x77 suppresses extension warnings for common extensions to Fortran 77 — these are TAB format, byte-length specifiers and Hollerith constants;

−w=x95 suppresses extension warnings for extensions to Fortran 95.

−W*options*

The *−W* option can be used to specify the path to use for a compilation component or to pass an option directly to such a component. The possible combinations are:

−W0=*path* Specify the path used for the Fortran Compiler front-end. Note that this does not affect the library directory; the *−Qpath* option should be used to specify that.

−Wc=*path* Specify the path to use for invoking the C compiler; this is used both for the final stage of compilation and for linking.

−Wc,*option* Pass *option* directly to the host C compiler when compiling (producing the .o file). Multiple options may be specified in a single *−Wc,* option by separating them with commas.

−Wl=*path* Specify the path to use for invoking the linker (producing the executable).

−Wl,*option* Pass *option* directly to the host C compiler when linking (producing the executable). Multiple options may be specified in a single *−Wl,* option by separating them with commas. A comma may be included in an option by repeating it, e.g. *−Wl,-filelist=file1,,file2,,file3* becomes the linker option *−filelist=file1,file2,file3*.

−Wp=*path* Specify the path to use for invoking the fpp preprocessor.

−Wp,*option* Pass *option* directly to fpp when preprocessing.

−wmismatch=*proc-name-list*

Specify a list of external procedures for which to suppress argument data type and arrayness consistency checking. The procedure names should be separated by commas, e.g. *−wmismatch=p_one,p2*. Unlike the *−mismatch* option, this only affects data type and arrayness checking, and no warning messages are produced.

−xlicinfo

Report on the availability of licences for the compiler instead of compiling anything.

−xs Store the symbol tables in the executable (otherwise debugging is only possible if the object files are kept).

6 Files

***file*.a** Library of object files.

***file*.c** C source file.

***file*.f** Fortran source file in fixed format (obsolete).

***file*.f90** Fortran source file in free format.

***file*.f95** Fortran source file in free format.

***file*.F** Preprocessor source file for fixed-form Fortran (obsolete).

***file*.ff90** Preprocessor source file for free-form Fortran.

***file*.F90** Preprocessor source file for free-form Fortran.

***file*.ff95** Preprocessor source file for free-form Fortran.

***file*.F95** Preprocessor source file for free-form Fortran.

***name*.mod** Compiled module information file; *name* is the name of the module in lower case.

***file*.o** Object file

/opt/NAG_Fortran/lib
Default NAG Fortran Compiler library directory on Sun Solaris (see *−Qpath*); referred to as *library* hereafter.

/usr/local/lib/NAG_Fortran
Default NAG Fortran Compiler library directory on other Unix-based operating systems.

C:\Program Files\NAG\EFBuilder 5.3\nagfor\lib
Default NAG Fortran Compiler library directory on 32-bit Windows.

C:\Program Files (x86)\NAG\EFBuilder 5.3\nagfor\lib
Default NAG Fortran Compiler library directory on 64-bit Windows.

***library*/f90_iostat.f90**
Source code for the `f90_iostat` module.

***library*/f90_kind.f90**
Source code for the `f90_kind` module.

***library*/f90_stat.f90**
Source code for the `f90_stat` module.

***library*/f90_util.f90**
A sample Fortran 90 program that displays implementation-specific information

***library*/iso_fortran_env.f90**
Source code for the `iso_fortran_env` module.

***library*/nagfmcheck.f90**
Source code for the *nagfmcheck* program, see the Memory Tracing section.

7 Compilation Messages

The messages produced by the NAG Fortran Compiler itself during compilation are intended to be self-explanatory. The linker, or more rarely the host C compiler, may produce occasional messages.

Messages produced by the compiler are classified by severity level; these levels are:

Info informational message, noting an aspect of the source code in which the user may be interested.

Warning the source code appears likely to be in error.

Questionable
some questionable usage has been found in the source code which may indicate a programming error. This has the same severity as "warning".

Extension some non-standard-conforming source code has been detected but has successfully been compiled as an extension to the language. This has the same severity as "warning".

Obsolescent
some archaic source code has been detected which although standard-conforming was classified as obsolescent by the Fortran 95 standard. This has the same severity as "warning".

Deleted feature used
a feature that was present in Fortran 90 but deleted from the Fortran 95 standard was used. This has the same severity as "warning".

Error the source code does not conform to the Fortran standard or does not make sense. Compilation continues after recovery.

Fatal a serious error in the user's program from which the compiler cannot recover, the compilation is immediately terminated.

Panic an internal inconsistency is found by one of the compiler's self-checks; this is a bug in the compiler itself and NAG should be notified.

8 Compiler Limits

Item	Limit
Maximum `INCLUDE` file nesting	20
Maximm number of `INCLUDE` file references per compilation	2047
Maximum `DO` loop nesting level	199
Maximum `CASE` construct nesting level	30
Maximum `DATA`-implied-`DO` loop nesting	99
Maximum array-constructor-implied-`DO` loop nesting	99
Maximum number of dummy arguments	32767
Maximum number of arguments to `MIN` and `MAX`	100
Maximum character length	2147483647
Maximum array size (32-bit systems)	2147483647 bytes
Maximum array size (64-bit systems)	64GB
Maximum unit number	2147483647
Maximum I/O record length	2147483647

9 Input/Output Information

Item	Value
Standard error (stderr) unit number	0
Standard input (stdin) unit number	5
Standard output (stdout) unit number	6
Default maximum record length for formatted output	1024 characters
Default maximum record length for unformatted output	2147483647 bytes

The default directory used for files opened with `STATUS='SCRATCH'` is '`/tmp`' on Unix and the Windows temporary directory on Windows. This default may be overridden with the `TMPDIR` environment variable.

10 OpenMP Support

The most commonly-used features of OpenMP 3.0 are supported. The following table describes the level of support for each OpenMP directive in the initial 5.3 release.

Executable directive	Level of support
PARALLEL	Supported except for the COPYIN clause.
DO	Supported except for COLLAPSE, LASTPRIVATE and ORDERED.
SECTIONS	Supported except for LASTPRIVATE.
SINGLE	Fully supported.
MASTER	Fully supported.
WORKSHARE	Not supported.
PARALLEL DO	Supported except as noted for PARALLEL and DO.
PARALLEL SECTIONS	Fully supported.
PARALLEL WORKSHARE	Not supported.
TASK	Not supported.
MASTER	Fully supported.
CRITICAL	Fully supported.
BARRIER	Fully supported.
TASKWAIT	Not supported.
ATOMIC	Fully supported.
FLUSH	Fully supported.
ORDERED	Not supported.
Data directive/clauses	**Level of support**
THREADPRIVATE	Not supported.
DEFAULT	Fully supported.
SHARED	Fully supported.
PRIVATE	Fully supported.
FIRSTPRIVATE	Supported in PARALLEL directives.
LASTPRIVATE	Not supported.
REDUCTION	Fully supported.
COPYIN	Not supported.
COPYPRIVATE	Not supported.

All the procedures in section 3.2 of the OpenMP standard are supported; these are omp_set_num_threads, omp_get_num_threads, omp_get_max_threads, omp_get_thread_num, omp_get_num_procs, omp_in_parallel, omp_set_dynamic, omp_get_dynamic, omp_set_nested, omp_get_nested, omp_set_schedule, omp_get_schedule, omp_get_thread_limit, omp_set_max_active_levels, omp_get_max_active_levels, get_level, omp_get_ancestor_thread_num, omp_get_team_size and omp_get_active_level.

The lock procedures in section 3.3 of the OpenMP standard are not supported in release 5.3.

The timing routines in section 3.4 of the OpenMP standard are supported; these are omp_get_wtime and omp_get_wtick.

When using the IEEE arithmetic support modules, the IEEE modes (rounding, halting and underflow) are propagated into spawned OpenMP threads at the beginning of a PARALLEL construct, and any IEEE flag that are set by an OpenMP thread is passed back to the parent thread at the end of the PARALLEL construct.

The following table lists the OpenMP environment variables with their default values and, if applicable, their limits.

Environment Variable	Default	Limits
OMP_NUM_THREADS	1	1-32768
OMP_DYNAMIC	False	true or false
OMP_NESTED	False	true or false
OMP_STACKSIZE	0	<1GB (32-bit) or 16GB (64-bit)
OMP_WAIT_POLICY	None	active or passive
OMP_MAX_ACTIVE_LEVELS	1	1-64
OMP_THREAD_LIMIT	32768	1-32768

Note that although the NAG runtime supports up to 32768 threads, operating system limits may prevent usage of so many.

OpenMP is not compatible with the *−C=undefined* option.

11 Automatic File Preconnection

All logical unit numbers are automatically preconnected to specific files. These files need not exist and will only be opened or created if they are accessed with `READ` or `WRITE` without an explicit `OPEN`. By default the specific filename for unit n is `fort.`n; however if the environment variable `FORT`nn exists its value is used as the filename. Note that there are two digits in this variable name, e.g. the variable controlling unit 1 is `FORT01` whereas the default filename is '`fort.1`' (unless the prefix has been changed, see the description of module `F90_PRECONN_IO`).

A file preconnected in this manner is opened with `ACCESS='SEQUENTIAL'`. If the initial `READ` or `WRITE` is an unformatted i/o statement, it is opened with `FORM='UNFORMATTED'` otherwise it is opened with `FORM='FORMATTED'`. By default a formatted connection is opened with `BLANK='NULL'` and `POSITION='REWIND'` (see module `F90_PRECONN_IO`).

Automatic preconnection applies only to the initial use of a logical unit; once `CLOSE`d the unit will not be reconnected automatically but must be explicitly `OPEN`ed.

Note that this facility means that it is possible for a `READ` or `WRITE` statement with an `IOSTAT=` clause to receive an i/o error code associated with the implicit `OPEN`.

12 IEEE 754 Arithmetic Support

If no floating-point option is specified, any floating divide-by-zero, overflow or invalid operand exception will cause the execution of the program to be terminated (with an informative message and usually a core dump). Occurrence of floating underflow may be reported on normal termination of the program. On hardware supporting IEEE 754 standard arithmetic gradual underflow with denormalised numbers will be enabled. Note that this mode of operation is the only one available on hardware which does not support IEEE 754.

If the *−ieee=full* option is specified, non-stop arithmetic is enabled; thus `REAL` variables may take on the values +Infinity, −Infinity and NaN (Not-a-Number). If any of the floating exceptions listed above are detected by the hardware during execution, this fact will be reported on normal termination. The *−ieee=full* option must be specified when compiling the main program and has global effect.

If the *−ieee=nonstd* option is specified, floating-point exceptions are handled in the default manner (i.e. execution is terminated). However, gradual underflow is **not** enabled, so results which would have produced a denormalised number produce zero instead. This option can only be used on hardware for which this mode of operation is faster. Like *−ieee=full*, the *−ieee=nonstd* option must be specified when compiling the main program and has global effect.

13 Random Number Algorithm

The random number generator supplied as the intrinsic subroutine `RANDOM_NUMBER` is the "Mersenne Twister".

14 Automatic Garbage Collection

The *−gc* option enables use of the runtime garbage collector. It is necessary to use this option during the link phase for it to have effect; specifying it additionally during the compilation phase can result in improved performance.

The supplied Technical Information note (TECHINFO) lists whether garbage collection is available for your system. If it is available, there will be a file '`gc.o`' in the compiler's library directory.

The collector used is based on version 5.3 of the publicly available general purpose garbage collecting storage allocator of Hans-J Boehm, Alan J Demers and Xerox Corporation, described in "Garbage Collection in an Uncooperative Environment" (H Boehm and M Weiser, Software Practice and Experience, September 1988, pp 807-820).

The copyright notice attached to their latest version is as follows:

Note that the "NO WARRANTY" disclaimer refers to the original copyright holders Boehm, Demers, Xerox Corporation, Silicon Graphics and Hewlett-Packard Company. The modified collector distributed in binary form with the NAG Fortran Compiler is subject to the same warranty and conditions as the rest of the NAG Fortran compilation system.

The module `F90_GC` is provided; it contains functions and variables that can control the behaviour of the garbage collector.

15 Memory Tracing

Tracing of memory allocation and deallocation is provided by the *−mtrace* option. Control is provided over whether the address, size, and line number of each allocation is displayed, or the tracing output can be suppressed entirely. A "paranoia" mode is provided where the memory allocator protects its data structures against inadvertent modification by the user program.

Runtime environment variables may be used to override the tracing options a program was built with, and to specify where to write the tracing output. These are only operative if the program was built with some tracing option; *−mtrace=off* will build a program with the tracing-capable memory allocator.

If *−mtrace=off* is not specified, use of any *−mtrace* option will implicitly do a *−mtrace=on*.

Basic tracing produces a message to the memory tracing file (normally standard error) for each allocation and deallocation, including those for automatic variables, i/o buffers and compiler-generated temporaries. Each allocation is numbered sequentially; the first three items are the i/o buffers for units 0, 5 and 6 (standard error, standard input and standard output).

All *−mtrace=* suboptions may be overridden at run time by the `NAGFORTRAN_MTRACE_OPTIONS` environment variable, which should be set to the required *trace_opt_list* (e.g. '`on,size`'). The memory tracing file may be specified at run time by the `NAGFORTRAN_MTRACE_FILE` environment variable.

The *−mtrace* option must be specified when linking, and is incompatible with *−gc*. Additionally, line number information is only available for those files compiled with *−mtrace=line*.

The `nagfmcheck` program can be used to check the output from the *−mtrace* option. It is designed to be used as a filter. Any lines that do not look like memory tracing output are ignored. It reports to standard output any errors it detects such as deallocating something twice, deallocating something that was never allocated, or deallocating something with a size different from that with which it was allocated. It also reports any apparent memory leaks, though this is less useful if the program terminated prematurely.

16 Undefined Variable Detection

Use of undefined variables can be detected with the *−C=undefined* option. Program units compiled with this option are incompatible with program units compiled without this option (i.e. the whole program must be compiled the same way). For this reason, *−C=undefined* is not part of *−C* or *−C=all*.

Currently, there are a number of other limitations on the use of *−C=undefined*.

1. It is incompatible with pointers in an initialised `COMMON`.
2. All intrinsic modules are available, but the `ISO_C_BINDING` module can only be used with all-Fortran programs as the option makes changes to the ABI.
3. Internal `READ` from a `CHARACTER` array requires the entire specified array subobject to be "defined", even those elements corresponding to records not actually read.
4. Internal `WRITE` to a `CHARACTER` array is considered to define the entire specified array subobject, even those elements corresponding to records not actually written.
5. Certain intrinsic functions require the entirety of their arguments to be defined, even if some portions are not actually required for the value of the function. For example, the `PAD` argument to `RESHAPE` when no padding is actually required, and elements of the `ARRAY` argument to `PACK` that correspond to false elements of the `MASK`.
6. It is incompatible with the use of OpenMP directives.

17 Data Types

The table below lists the data types provided by the NAG Fortran Compiler together with their kind numbers. There are two possibilities for the KIND numbers: the default mode of operation (which may be specified explicitly by the *−kind=sequential* option) and the "byte" numbering scheme (specified by the *−kind=byte* option).

Type Name	KIND Number (kind=sequential)	KIND Number (kind=byte)	Description
REAL	1	4	Single precision floating-point
REAL	2	8	Double precision floating-point
REAL	3	16	Quadruple precision floating-point
COMPLEX	1	4	Single precision complex
COMPLEX	2	8	Double precision complex
COMPLEX	3	16	Quadruple precision complex
LOGICAL	1	1	Single byte logical
LOGICAL	2	2	Double byte logical
LOGICAL	3	4	Default logical
LOGICAL	4	8	Eight byte logical
INTEGER	1	1	8-bit integer
INTEGER	2	2	16-bit integer
INTEGER	3	4	32-bit (default) integer
INTEGER	4	8	64-bit integer
CHARACTER	1	1	ASCII or ISO 8859-1 character
CHARACTER	2	2	JIS X 0213 character
CHARACTER	3	3	Unicode (UCS-2) character
CHARACTER	4	4	ISO 10646 (UCS-4) character

Note that on all machines except Sun Solaris with the SunPro C compiler, quadruple precision is actually "double double" precision; this provides nearly twice the precision of Double precision but with a reduced exponent range.

The `F90_KIND` module contains named parameters useful for specifying which kind you want regardless of whether the numbering system is "sequential" or "byte".

Additional intrinsic modules built into the NAG Fortran Compiler are described in the "nag_modules" document.

18 Modules

To use a module it must be previously compiled, or must be defined in the file prior to its use. When separately compiling a module the *−c* option should be specified.

Compiling a module creates a '.mod' file and a '.o' file. The '.mod' file is used by the compiler at compile time to provide information about module contents, the '.o' file (if generated) contains the code of any module procedures and must be specified when creating an executable file.

Note that the name of the '.mod' file will be the name of the module, the '.o' file will be named after the original source file.

When a previously compiled module is USEd the NAG Fortran Compiler attempts to find its source file and, if that is successful, checks the modification times producing a warning message if the '.mod' file is out of date.

19 Runtime Environment Variables

The following variables control the runtime environment for programs compiled with the NAG Fortran Compiler.

`NAGFORTRAN_MTRACE_FILE`
: Programs compiled using any *−mtrace=* option will write the memory trace to this file. The default is standard error.

`NAGFORTRAN_MTRACE_OPTIONS`
: Changes the memory tracing options for programs compiled using any *−mtrace=* option.

`NAGFORTRAN_RUNTIME_ERROR_FILE`
: Runtime error messages will be written to this file. The default is standard error.

`NAGFORTRAN_RUNTIME_LANGUAGE`
: Controls the language used for runtime error messages. This may be 'English' or 'Japanese' (not case-sensitive); the default is English.

`TMPDIR` Controls the directory used for scratch files (the default is system-dependent).

20 Debugging

For operating systems other than Windows a Modern Fortran-aware debugger might be available as dbx90; see TECHINFO.txt for details.

In general, host system debuggers, such as dbx or gdb, may be used successfully on Fortran code as the names of the original source files, plus line numbers, are passed through to the intermediate C files. In using such debuggers it should be noted that most local variables have an underscore appended to their names. It may be useful to look at the intermediate C code when debugging; this is produced by the *−S* option.

21 Producing a Call Graph

The call graph generator takes a set of Fortran source files and produces a call graph with optional index and called-by tables. C files and fpp-processed files are not handled.

The call graph generator understands the following compiler options with the same meaning: *−132*, *−compatible*, *−dcfuns*, *−double*, *−dryrun*, *−dusty*, *−encoding*, *−english*, *−f2003*, *−f2008*, *−f95*, *−fixed*, *−free*, *−help*, *−hollerith_io*, *−hpf*, *−I*, *−indirect*, *−info*, *−kind*, *−maxcontin*, *−mismatch*, *−mismatch_all*, *−nihongo*, *−nocheck_modtime*, *−nomod*, *−o*, *−Qpath*, *−r8*, *−strictf95*, *−u*, *−v*, *−V*, *−w* and *−xlicinfo*.

The "@*filename*" syntax may also be used, with the same effect as the "*−indirect filename*" option.

The call graph is written to the file specified by the *−o* option, or to standard output if no *−o* option is specified.

The following additional options control the output produced.

−calledby
: Produce a "called-by" table showing, for each routine, the routines that call it directly or indirectly. This is produced at the end of the output.

–indent=*N*
Indent by *N* for each level in the graph, up to the maximum. The default is *–indent=4*.

–indent_max=*N*
The maximum indentation is *N*. The default is *–indent_max=70*.

–index Produce an alphabetic index listing, for each routine, the line of the call graph where the routine first appears. This follows the call graph itself and precedes the called-by table (when the *–calledby* option is used).

–show_entry
Show `ENTRY` point names in the call graph; without this option, calls to an `ENTRY` point are shown as calls to the containing subprogram.

–show_generic
If a call is via a generic identifier, show the generic identifier in the call graph.

–show_host
Show the host scope names for calls to internal and module procedures.

–show_pclass
Show the class of each procedure (e.g. 'module', 'internal', ...).

–show_rename
If a called procedure was renamed on a `USE` statement, show the renaming.

22 Dependency Analysis

The dependency analyser takes a set of Fortran source files and produces dependency information in the form specified. C files and fpp-processed files are not handled.

The dependency analyser understands the following compiler options with the same meaning: *–132*, *–dryrun*, *–english*, *–fixed*, *–free*, *–help*, *–hpf*, *–I*, *–indirect*, *–nihongo*, *–o*, *–Qpath*, *–tempdir*, *–v* and *–V*. The "@*filename*" syntax may also be used with the same effect as the "*–indirect filename*" option.

The output form is controlled by the *–otype=***type** option, where **type** is one of:

blist (the filenames as an ordered build list),
dfile (the dependencies in Makefile format, written to separate *file*`.d` files),
info (the dependencies as English descriptions) or
make (the dependencies in Makefile format).

The default is *–otype=info*. If *–otype=dfile* is specified, no *–o* option is permitted; otherwise, the result is written to the file specified by the *–o* option or to standard output if no *–o* option is specified.

23 Source File Polishing

The polisher takes a set of Fortran source files, which may be in fixed or free form, and produces a free form "polished" version of each file. C files and fpp-processed files are not handled.

The polisher understands the following compiler options with the same meaning: *–132*, *–encoding*, *–english*, *–fixed*, *–free*, *–help*, *–I*, *–indirect*, *–info*, *–nihongo*, *–noqueue*, *–o*, *–Qpath*, *–tempdir*, *–u*, *–v*, *–V*, *–w* and *–xlicinfo*.

The polished output is written to the file specified by the *–o* option, or to the same filename with the extension replaced by '.f90_pol' if no *–o* option is specified. The output file cannot have the same name as the input file.

The following additional options control the operation of the polisher:

–alter_comments
Enable options to alter comments; without this option, any options that would otherwise alter the comments are ignored.

–blank_cmt_to_blank_line
Turn comment lines that have no text (other than the comment-initiating character) into plain blank lines; this is the default if the *–alter_comments* option is set.

–blank_line_after_decls
Ensure that there is a blank line after the declarations and before the first executable statement; this is the default.

–bom=*X*
Specify whether to write a Unicode Byte-Order Mark at the beginning of the output file; *X* must be one of **Asis** (same as the input file), **Insert** (insert a byte-order mark) or **Remove** (remove any byte-order mark). This option only has effect if the input file is known to be in UTF-8 encoding, either because it begins with a byte-order mark or the *–encoding=UTF8* option was used. The default is *–bom=Asis*.

–break_long_comment_word
If a comment line will be split into two lines, the comment may be broken in the middle of a long word.

–delete_all_comments
Delete all comments (if the *–alter_comments* option is set).

–delete_blank_lines
Delete blank lines and comment lines that have no text (other than the comment-initiating character), if the *–alter_comments* option is set.

–delete_unused_labels
Delete labels that are never referenced; this is the default.

–format_start=*N*
If renumbering `FORMAT` statements in a separate sequence, the first `FORMAT` statement will be *N*; the default is *–format_start=90000*.

–format_step=*N*
If renumbering `FORMAT` statements in a separate sequence, the step from one label to the next will be *N*; the default is *–format_step=10*. Note that this may be negative (but not zero).

–idcase=*X*
Set the case to use for identifiers; *X* must be one of **C** (for Capitalised), **L** (for lowercase) or **U** (for UPPERCASE); the default is *–idcase=L*.

–indent=*N*
Indent statements within a construct by *N* spaces from the current indentation level; the default is *–indent=2*.

–indent_comment_marker
When indenting comments, the comment-initiating character should be indented to the indentation level; this is the default.

–indent_comments
Indent comments; this is the default if the *–alter_comments* option is set. The result is also affected by the *–indent_comment_marker* option.

–indent_continuation=*N*
Indent continuation lines by an additional *N* spaces; the default is *–indent_continuation=2*.

–indent_max=*N*
Set the maximum indentation level to *N* spaces; the default is *–indent_max=60*. The value must be at least 10 less than the output line length (*–width=*).

–inline_comment_indent=*N*
Set the indentation level for inline comments to column *N*; the default is *–inline_comment_index=35*.

–keep_blank_lines
Do not delete blank lines or comment lines with no text; this is the opposite of *–delete_blank_lines* and is the default.

–keep_comments
Do not delete non-blank comment lines; this is the opposite of *–delete_comments* and is the default.

–kind_keyword=*X*
Specifies how to handle the `KIND=` specifier in declarations; *X* must be one of **Asis** (take no action but preserve the input status), **Insert** (insert `KIND=` if not present), or **Remove** (remove `KIND=` if present); the default is *–kind_keyword=Asis*.

–kwcase=*X*
Set the case to use for language keywords; *X* must be one of **C** (for Capitalised), **L** (for lowercase) or **U** (for UPPERCASE); the default is *–kwcase=C*.

–label_after_indent
Indent labels; this is the opposite to *–label_before_indent*.

–label_before_indent
Output the statement label, if any, before indenting the statement; this is the default.

–leave_formats_in_place
Leave `FORMAT` statements in the same position as they are in the input file; this is the opposite of *–move_formats_to_end*, and is the default.

–margin=*N*
Set the left margin (initial indent) to *N*. The value must be at least 10 less than the output line length (*–width=*).

–name_scopes=*X*
Specify whether to add optional keywords and scope names to the `END` or `END TYPE` statement for a scope; *X* must be one of **Asis** (leave as is), **Insert** (insert keywords and/or names) or **Remove** (remove optional keywords and names). This option also applies to the `END INTERFACE` statement. The default is *–name_scopes=Insert*.

–noalter_comments
Do not alter comments in any way; this is the default.

–noblank_cmt_to_blank_line
Do not turn blank comments to blank lines.

–noblank_line_after_decls
Do not insert a blank line between the last declaration and the first executable statement.

–nobreak_long_comment_word
If a comment line will be split into two lines, do not break the comment in the middle of a long word; this is the default.

–noindent_comment_marker
Place the comment-initiating character for a comment line in column 1.

–noindent_comments
Indent the text of a comment line.

–norenumber
Do not renumber statement labels.

–noseparate_format_numbering
When renumbering statement labels, use a single sequence for both `FORMAT` and non-`FORMAT` statements; this is the default.

–noterminate_do_with_enddo
Do not change `DO` loop terminating statements.

–relational=*X*
Specifies the form to use for relational operators, *X* must be either **F77-** (use `.EQ.`, `.LE.`, etc.) or **F90+** (use `==`, `<=`, etc.); the default is *–relational=F90+*.

–renumber
Renumber statement labels; this is the default.

–renumber_start=*N*
When renumbering statement labels, the first label will be *N*; the default is *–renumber_start=100*.

–renumber_step=*N*
When renumbering statement labels, the step from one label to the next will be *N*; the default value is *–renumber_step=10*.

–separate_format_numbering

When renumbering statement labels, renumber `FORMAT` statements in a separate sequence from non-`FORMAT` statements.

–terminate_do_with_enddo

Change the terminating statements of all `DO` loops so that each loop ends with an `ENDDO` statement; this is the default.

–width=*N*

Set the maximum length of the text on each output line to N; the default is *–width=78*. Note that in the case of continuation lines, an additional two characters (‘ `&`’) will be produced after the last text on a line and this may take the line length over the limit. The width must be at least 10 more than the left margin (*–margin=*) and the maximum indent (*–indent_max=*). The maximum width setting is 1024, however values higher than 130 will produce output that does not conform to the Fortran standard.

24 See Also

f90_gc(3), **f90_iostat**(3), **f90_kind**(3), **f90_preconn_io**(3), **f90_stat**(3), **f90_unix_dir**(3), **f90_unix_dirent**(3), **f90_unix_env**(3), **f90_unix_errno**(3), **f90_unix_file**(3), **f90_unix_proc**(3), **ieee_arithmetic**(3), **ieee_exceptions**(3), **ieee_features**(3), **iso_c_binding**(3), **iso_fortran_env**(3), **nag_modules**(3), **nagfmcheck**(1).

25 Bugs

Please report any bugs found to ‘support@nag.co.uk’ or ‘support@nag.com’, along with any suggestions for improvements.

26 Author

Malcolm Cohen, Nihon Numerical Algorithms Group KK, Tokyo, Japan.