NAG Fortran Compiler Release 5.3 Release Note

July 31, 2012

1 Introduction

Release 5.3 of the NAG Fortran Compiler contains many improvements to the compiler and additions to the Fortran 2003 and Fortran 2008 language features.

Licence keys for Releases 5.1 and 5.2 will not work with Release 5.3; contact NAG to obtain a new licence key.

See `KLICENCE.txt` for more information about Kusari Licence Management.

1.1 Compatibility with Release 5.2

Release 5.3 of the NAG Fortran Compiler is fully compatible with Release 5.2.

1.2 Compatibility with Release 5.1

Release 5.2 of the NAG Fortran Compiler is compatible with NAGWare f95 Release 5.1 except that:

- programs or libraries that use the `CLASS` keyword, or which contain types that will be extended, need to be recompiled;
- the following 64-bit platforms, when the *−abi=64* option (the default) is used, are binary incompatible and all programs and libraries need to be recompiled: NPL6A51NA, NPMI651NA.

1.3 Compatibility with Earlier Releases

Except as noted, the NAG Fortran Compiler release 5.3 is compatible with NAGWare f90 Releases 2.1 and 2.2, as well as with all NAGWare f95 Releases from 1.0 to 5.0, except as noted below.

The following incompatibilities were introduced in Release 5.1:

- The value returned by `STAT=`, on an `ALLOCATE` or `DEALLOCATE` statement, may differ from the pre-5.1 value in some cases. For further information see the `F90_STAT` module documentation.
- Programs that used type extension (`EXTENDS` attribute) in 5.0 need to be recompiled.
- Formatted output for IEEE infinities and NaNs is different, and now conforms to Fortran 2003.
- List-directed output of a floating-point zero now uses F format, as required by Fortran 2003, instead of E format.
- An i/o or format error encounted during `NAMELIST` input will now skip the erroneous record. This behaviour is the same as all other formatted input operations including list-directed.

1.4 New Features

Release 5.3 supports OpenMP 3.0 (only when the *−openmp* option is used) and includes many new features from the Fortran 2003 and Fortran 2008 standards.

This release also contains performance enhancements and other minor enhancements. Additionally, several programming tools have been integrated into the compiler system.

2 OpenMP 3.0 Support

Support for the most commonly-used features of OpenMP 3.0 has been added. In the initial 5.3 release, support for each directive is described in the table below.

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.
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 this release.

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

All OpenMP environment variables are supported.

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.

3 Fortran 2003 Features

- `FINAL` subroutines have been added, to provide user-specified finalisation.
- Structure constructors can now use keywords for component selection, and can omit values for components that have default initialisation. Using keywords it is possible to provide a single value for the whole parent component (or another ancestor component) instead of each inherited component individually.
- Structure constructors can be overloaded with generic functions.
- The intrinsic function `EXTENDS_TYPE_OF`.
- The intrinsic function `SAME_TYPE_AS`.
- Any kind of `LOGICAL` variable is now accepted in the `INQUIRE` statement for the `EXIST=`, `NAMED=`, `OPENED=` and `PENDING=` specifiers.
- The intrinsic type parameter enquiries `%KIND` and `%LEN` have been added; these can be applied to variable designators. The form *variable*`%KIND` is the same as `KIND(`*variable*`)`, and *variable*`%LEN` is the same as `LEN(`*variable*`)`, except that the enquiry forms can be used even if the intrinsic function names are not available. Also, the `%LEN` form returns the length in a kind that is big enough to hold the length, this is not necessarily the same as default integer.
- All intrinsic functions are now allowed in constant expressions.
- The full ISO 10646 4-byte character set is supported; the kind is returned by the intrinsic function reference `SELECTED_CHAR_KIND('ISO_10646')`. As ISO 10646 defines a 2-byte subset (UCS-2) as well as the full 4-byte characters (UCS-4), the intrinsic function `SELECTED_CHAR_KIND` also recognised the name `'UCS_4'` for this character set.
- The ISO 10646 file encoding UTF-8 is supported; to use this, open the file with `ENCODING='UTF-8'`. Any kind of character can be written to and read from a UTF-8 file.
- The intrinsic module `ISO_FORTRAN_ENV` contains the additional scalar integer constant `IOSTAT_INQUIRE_INTERNAL_UNIT`. A later release of the NAG Fortran Compiler will support defined input/output; at that time, the value of `IOSTAT_INQUIRE_INTERNAL_UNIT` identifies the `IOSTAT=` error value that will result from using `INQUIRE` in a child input/output statement for an internal file.

4 Fortran 2008 Features

- The `BLOCK` construct has been added; this construct allows declarations of entities within executable code. For example,

```
Do i=1,n
  Block
    Real tmp
    tmp = a(i)**3
```

```
    If (tmp>b(i)) b(i) = tmp
  End Block
End Do
```

Here the variable `tmp` has its scope limited to the `BLOCK` construct, so will not affect anything outside it. This is particularly useful when including code by `INCLUDE` or by macro preprocessing.

All declarations are allowed within a `BLOCK` construct except for `COMMON`, `EQUIVALENCE`, `IMPLICIT`, `INTENT`, `NAMELIST`, `OPTIONAL` and `VALUE`; also, statement function definitions are not permitted. `BLOCK` constructs may be nested; like other constructs, branches into a `BLOCK` construct from outside are not permitted.

- In a structure constructor, the value for an allocatable component may be omitted: this has the same effect as specifying `NULL()`.
- In a `STOP` statement, the *stop-code* may be any scalar constant expression of type integer or default character. (This also applies to the `PAUSE` statement, but that statement is no longer standard Fortran.)
- When the *−f2008* option is in effect, `ENTRY` statements will be reported as obsolescent.
- The intrinsic module `ISO_FORTRAN_ENV` contains the additional scalar integer constants `INT8`, `INT16`, `INT32`, `INT64`, `REAL32`, `REAL64` and `REAL128`; these supply the kind type parameter values for integer and real kinds with the indicated bit sizes.
- The intrinsic module `ISO_FORTRAN_ENV` contains four additional named constants that are arrays: `CHARACTER_KINDS`, `INTEGER_KINDS`, `LOGICAL_KINDS` and `REAL_KINDS`; these list the available kind type parameter values for each type (in no particular order).
- An empty internal subprogram part, module subprogram part or type-bound procedure part is now permitted following a `CONTAINS` statement. In the case of the type-bound procedure part, an ineffectual `PRIVATE` statement may appear following the unnecessary `CONTAINS` statement.
- A type-bound procedure declaration statement may now declare multiple type-bound procedures. For example, instead of

```
PROCEDURE,NOPASS :: a
PROCEDURE,NOPASS :: b=>x
PROCEDURE,NOPASS :: c
```

  the single statement

```
PROCEDURE,NOPASS :: a, b=>x, c
```

  will suffice.

- The `NEWUNIT=` specifier has been added to the `OPEN` statement; this allocates a new unit number that cannot clash with any other logical unit (the value will be a special negative value). For example,

```
INTEGER unit
OPEN(FILE='output.log',FORM='FORMATTED',NEWUNIT=unit)
WRITE(unit,*) 'Logfile opened.'
```

  The `NEWUNIT=` specifier can only be used if either the `FILE=` specifier is also used, or if the `STATUS=` specifier is used with the value `'SCRATCH'`.

- The elemental intrinsic functions `BGE`, `BGT`, `BLE` and `BLT` have been added; these do bitwise (i.e. unsigned) comparisons. They have two arguments, `I` and `J`, which must be of type Integer but may be of different kind. The result is default Logical.

  For example, `BGE(INT(Z'FF',INT8),128)` is true, while `INT(Z'FF',INT8)>=128` is false.

- The array reduction intrinsic functions `IALL`, `IANY` and `IPARITY` have been added. These are exactly the same as `SUM` and `PRODUCT`, except that instead of reducing the array by the `+` or `*` operation, they reduce it by the `IAND`, `IOR` and `IEOR` intrinsic functions respectively. That is, each element of the result is the bitwise-and, bitwise-or, or bitwise-exclusive-or of the reduced elements. If the number of reduced elements is zero, the result is zero for `IANY` and `IPARITY`, and `NOT(`*zero*`)` for `IALL`.

- The elemental intrinsic functions `LEADZ` and `TRAILZ` have been added; these return the number of leading (most significant) and trailing (least significant) zero bits in the argument `I`, which must be of type Integer (of any kind). The result is default Integer.

- The elemental intrinsic functions `MASKL` and `MASKR` have been added; these generate simple left-justified and right-justified bitmasks. The value of `MASKL(I,KIND)` is an integer with the specified kind that has its leftmost `I` bits set to one and the rest set to zero; `I` must be non-negative and less than or equal to the bitsize of the result. If `KIND` is omitted, the result is default integer. The value of `MASKR` is similar, but has its rightmost `I` bits set to one instead.

- The array reduction intrinsic function `PARITY` has been added. It is exactly the same as `ALL` and `ANY`, except that instead of reducing the array by the `.AND.` or `.OR.` operation, it reduces it by the `.NEQV.` operation. That is, each element of the result is `.TRUE.` if an odd number of reduced elements is `.TRUE.`.

- The elemental intrinsic functions `POPCNT` and `POPPAR` have been added. `POPCNT(I)` returns the number of bits in the Integer argument `I` that are set to 1. `POPPAR(I)` returns zero if the number of bits in `I` that are set to 1 are even, and one if it is odd. The result is default Integer.

- The Fortran 2008 rules are used for generic resolution and for checking that procedures in a generic are unambiguous. Under these rules,

  – a dummy procedure is distinguishable from a dummy variable;

  – an `ALLOCATABLE` dummy variable is distinguishable from a `POINTER` dummy variable that does not have `INTENT(IN)`.

5 Performance Enhancements

- The intrinsic functions `ACHAR` and `CHAR` are now faster in some cases, in particular when applied to arrays or when used in the middle of an expression.

- The intrinsic functions `ADJUSTL`, `ADJUSTR`, `LEN_TRIM` and `TRIM` are now faster in many cases.

- The intrinsic functions `MAXLOC` and `MINLOC` are faster on character strings.

- The intrinsic function `MERGE` is now faster in some cases, in particular when the non-chosen source expression would have been expensive to compute.

- The intrinsic function `SPREAD` is now faster on contiguous arrays.

- The intrinsic function `TRANSPOSE` is now faster on large arrays, in particular ones that do not fit in cache.

- Some variable-sized array constructors are evaluated more quickly, in particular ones which have many elements and an implied-DO structure that is complicated.
- The performance of the memory allocator when the *−thread_safe* option is used has been improved.
- Performance of character assignment has been improved in many cases.

6 Additional Error Checking

- Many-one array assignment, where the left-hand side of an intrinsic assignment statement is a vector-subscripted array section and the vector subscript has duplicate values, is now detected as an error at compilation time when the vector is constant. When the vector is not constant, this error will be detected at runtime if the *−C=array* option is used.
- More errors in pointer usage are detected at compile time.
- `IEEE_SET_ROUNDING_MODE` now detects a bad `ROUND_VALUE` argument. `IEEE_VALUE` now detects a bad `CLASS` argument.
- Shape mismatch is more reliably detected in elemental subroutine calls.
- More shape mismatch errors in intrinsic functions are detected at compile time.

7 Other Enhancements

- The intrinsic module `ISO_C_BINDING` is available with the *−C=undefined* option. Note that this option changes the ABI in a way that is incompatible with C; however, it is usable in an all-Fortran program.
- The standard intrinsic modules for IEEE arithmetic support, `IEEE_ARITHMETIC`, `IEEE_EXCEPTIONS` and `IEEE_FEATURES`, are now available with the *−C=undefined* option.
- The NAG intrinsic modules `F90_GC` and `F90_PRECONN_IO` are now available with the *−C=undefined* option.
- The NAG intrinsic modules for POSIX support, `F90_UNIX_DIR`, `F90_UNIX_DIRENT`, `F90_UNIX_ENV`, `F90_UNIX_ERRNO`, `F90_UNIX_FILE`, `F90_UNIX_IO`, `F90_UNIX_PROC` and `F90_UNIX`, are now available with the *−C=undefined* option.
- Improved details in error messages for misusing `INTENT(IN)` pointers.
- `DO` loops that cannot loop because of a `STOP` or `EXIT` statement are detected and a warning produced.
- Line number information has been added to some additional runtime errors when the *−g* option is used.
- The *−abi=64* and *−abi=32* options can now be used on Windows x64 to generate 64-bit and 32-bit programs.
- The *−C=recursion* option can now be used at the same time as the *−thread_safe* option.
- The *−no_underflow_warning* option has been added. This link-time option suppresses the warning message that normally appears if a floating-point underflow occurred during execution.

- The CONVERT= specifier has been added to the OPEN statement. Acceptable values are 'BIG_ENDIAN', 'BIG_IEEE', 'BIG_IEEE_DD', 'BIG_NATIVE', 'LITTLE_ENDIAN', 'LITTLE_IEEE', 'LITTLE_IEEE_DD', 'LITTLE_NATIVE', and 'NATIVE'. This is only allowed for unformatted files, and enables runtime conversion of endianness and file format. Conversion can also be enabled by the environment variable FORT_CONVERT*n*, where *n* is the unit number of the file to be converted; this takes precedence over any CONVERT= specifier on the OPEN statement. The *−convert=* option can also be used to set the default conversion mode for all units that are connected with no CONVERT= specifier or environment variable.
- Underflow control via IEEE_SET_UNDERFLOW_MODE (and IEEE_GET_UNDERFLOW_MODE) is now available on Windows x64, x86-64 Linux and Apple Intel Mac.
- Warning messages are now produced for variables that are set but which are definitely never referenced. These warnings can be disabled with the *−w=unreffed* option.
- Text files written in CRLF format (DOS/Windows) can now be read as formatted files by a Fortran program running on Unix or Linux.
- The JIS X 0213:2004 Japanese character set is supported; the kind is returned by the intrinsic function reference SELECTED_CHAR_KIND('JIS_0213'). As this character set is also sometimes referred to as 'Shift-JIS', the name 'SHIFT_JIS' is also recognised by the intrinsic function SELECTED_CHAR_KIND.
- The 2-byte Unicode character set (UCS-2) is supported; the kind is returned by the intrinsic function reference SELECTED_CHAR_KIND('UCS_2').
- The Shift-JIS file encoding is supported; to use this, open the file with ENCODING='SHIFT_JIS'. Any kind of character can be written to and read from a Shift-JIS file. When writing an ISO-8859-1, UCS-2 or UCS-4 character, if it cannot be represented in Shift-JIS it will be changed to a question mark ('?'). Similarly, when reading into an ISO-8859-1, UCS-2 or UCS-4 variable, if the character read cannot be represented in the variable's character set it will be changed to question mark ('?') if the variable is ISO-8859-1 and to the Unicode replacement character CHAR(INT(Z'fffd'),KIND(*variable*)) if the variable is UCS-2 or UCS-4.
- The F90_KIND module now contains the named constants JIS, UCS2 and UCS4 as kind selectors for the Japanese, Unicode UCS-2 and ISO 10646 UCS-4 character sets.
- Write-only files are now supported. When a write-only file is OPENed with POSITION='ASIS' (or no POSITION= specifier), the file is positioned at the end; this differs from other files which are positioned at the beginning. If the file can be positioned, the REWIND statement will set the position to the beginning; however, the BACKSPACE statement is not operational.
- The INQUIRE statement now reports more accurate information, in particular for the SIZE=, READ=, READWRITE= and WRITE= specifiers, for both connected and unconnected files.
- Integer literal constants that are 1 bigger than the maximum value for the particular integer kind, but which are preceded by a unary negation, are accepted. This allows, for example, the expression -128_1 which strictly speaking should be written -127_1-1_1; the former is clearer and more convenient.
- Allow BIND(C) procedures to be ELEMENTAL. References to such procedures will be done in array element order.

8 Integrated Programming Tools

Three software tools that operate on Fortran source files have been integrated into the NAG compiler system.

- Call Graph Generator — produces a call graph, with optional index and called-by tables.

- Dependency Analyser — produces information about dependencies on modules or `INCLUDE` files, in 'make' format, as English prose, or as an ordered build list.
- Polish — polishes ("pretty-prints") Fortran source files, optionally renumbering statement labels and converting `DO` statements to the block `DO` form.

9 Miscellaneous

- On Linux, Mac and Unix systems, the *−g* option no longer generates `upsf95` debugging information as this debugger has been withdrawn. On these platforms the *−g* option is now simply passed to the C compiler.
- A new option, *−max_parameter_size=N*, now controls the maximum size (in MB) allowed for a named constant (`PARAMETER`) at compile time. The size of named constants is limited to avoid machine slowdowns if a stupendously enormous constant is created accidentally. The default maximum size is 50 MB.
- The old *−f77* option has been renamed *−compatible*, to better reflect its actual effect (ABI compatibility with another Fortran compiler) and to avoid confusion with *−f95*, *−f2003* and *−f2008* (these all set the Fortran standard level).

10 New Fortran Standard

The extensions (described above) which follow the rules of the Fortran 2003 and Fortran 2008 standards are listed below together with the appropriate section number for the reference book "Modern Fortran Explained" by Metcalf, Reid & Cohen, Oxford University Press, 2011 printing (ISBN 978-0-19-960142-4).

Section	**Feature**
13.3	Type parameter enquiries (`%KIND` and `%LEN`).
14.8	Final subroutines.
14.10	`EXTENDS_TYPE_OF` and `SAME_TYPE_AS`.
15.3	Structure constructor keywords and overloading.
15.10	Intrinsic functions in constant expressions.
17.13	`INQUIRE` statement `LOGICAL` variable kinds.
20.1.3	Improved type-bound procedure declaration statement.
20.1.4	Omitting an allocatable value in a structure constructor.
20.1.6	`STOP` statement changes.
20.5.7	Generic resolution changes.
20.7.2	The `NEWUNIT=` specifier.
20.10.1	New intrinsic functions `BGE`, `BGT`, `BLE` and `BLT`.
20.10.3	New intrinsic functions `IALL`, `IANY` and `IPARITY`.
20.10.4	New intrinsic functions `LEADZ`, `POPCNT`, `POPPAR` and `TRAILZ`.
20.10.5	New intrinsic functions `MASKL` and `MASKR`.
20.10.7	New intrinsic functions `SHIFTA`, `SHIFTL` and `SHIFTR`.
20.11.5	New intrinsic function `PARITY`.
20.12.2	Names in `ISO_FORTRAN_ENV` for common kinds.
20.12.3	Arrays of kinds in `ISO_FORTRAN_ENV`.
B.10.2	Redundant `CONTAINS` statement.
C.2	Obsolescent `ENTRY` statement.