1 Introduction

Release 5.2 of the NAG Fortran Compiler contains many improvements to the compiler and additions to the Fortran 2003 language features.

This release of the compiler uses the same licensing scheme as Release 5.1; thus Release 5.1 licence keys continue to work with Release 5.2.

See KLICENCE for more information about Kusari Licence Management.

1.1 Compatibility with Release 5.1

Release 5.2 of the NAG Fortran Compiler is fully 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.

1.2 Compatibility with Earlier Releases

Except as noted, the NAG Fortran Compiler release 5.2 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.

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 encountered during NAMELIST input will now skip the erroneous record. This behaviour is the same as all other formatted input operations including list-directed.

1.3 New Features

Release 5.2 includes new features from the Fortran 2003 standard, plus quad precision floating-point arithmetic support on all platforms, improved runtime error message reporting, and a better random number generator.

This release also contains performance enhancements and other minor enhancements.
2 Major Fortran 2003 Features

- Unlimited polymorphic ‘CLASS(*)’ dummy arguments, pointers, and allocatable variables.
- Procedure pointers.
- Object-bound procedures (also known as “procedure pointer components”).
- Scalar allocatable variables.
- Deferred character length (‘LEN=’); together with scalar allocatable variables this provides a true variable-length character facility.
- Allow many more intrinsic functions to appear in initialisation expressions, including: ABS (of real and complex), ACOS, AIMAG, AINT, ANINT, ASIN, ATAN, ATAN2, CEILING, CMPLX, COS, COSH, COUNT, DBLE, DIM (of real), EXP, EXPONENT, FLOOR, FRACTION, INDEX (with BACK= argument), LGE, LGT, LTE, LT, LOG, LOG10, LOGICAL, MAX (of non-integer), MAXLOC, MAXVAL, MERGE, MIN (of non-integer), MINLOC, MINVAL, MOD (of real), MODULO, NEAREST, NINT, REAL, RRSPACING, SCALE, SCAN (with BACK= argument), SIGN (of non-integer), SIN, SINC, SET_EXPONENT, SPACING, SQRT, TAN, TANH, VERIFY (with BACK= argument). Also, exponentiation (** with the second argument being real or complex is now allowed in an initialisation expression.
- Assigning to an unallocated allocatable variable, or to a whole allocatable array with the wrong shape, now allocates the variable to be the correct shape (previously this was just an error).
- Recursive input/output. The Fortran 2003 standard only allows recursive input/output when at most one input/output statement identifies an external unit: however, the NAG Fortran Compiler allows recursive input/output with multiple external units — the only restriction is that the same external unit not be involved.
- The ASSOCIATE construct.

3 Other Fortran 2003 Features

- The intrinsic subroutine MOVE_ALLOC.
- KIND= arguments have been added to the intrinsic functions COUNT, IACHAR, ICHAR, INDEX, LBOUND, LEN, LEN_TRIM, MAXLOC, MINLOC, SCAN, SHAPE, SIZE, UBOUND and VERIFY.
- The intrinsic functions MAX, MAXLOC, MAXVAL, MIN, MINLOC and MINVAL now accept arguments of type CHARACTER.
- The default maximum number of continuation lines is now 255.
- An array constructor may now begin with a type-spec (followed by a double colon); in this case no actual values are required (with no values it has zero size with the specified type/kind/length) and any values are converted to the specified type/kind/length.
- The FLUSH statement.
- Asynchronous input/output including the ASYNCHRONOUS attribute and the WAIT statement; this provides all the syntax and semantics of asynchronous input/output, but the actual input/output itself is still done synchronously.
- A complex literal constant may have a named real or integer constant as the real and/or imaginary part (strictly speaking, this is not a “literal” constant).
• The lower bounds of an array pointer may be specified in a pointer assignment statement.

• User-defined operators (e.g. ‘.FRED.’) may now be renamed on the USE statement.

• The function C_F_PROCPOINTER has been added to the intrinsic module ISO_C_BINDING.

• The functions IEEE_GET_UNDERFLOW_MODE, IEEE_SET_UNDERFLOW_MODE, IEEE_SUPPORT_IO and IEEE_SUPPORT_UNDERFLOW_CONTROL have been added to the intrinsic module IEEE_ARITHMETIC. Note that underflow control itself is not supported.

• The named constant IEEE_OTHER_VALUE has been added to the intrinsic module IEEE_ARITHMETIC.

• The ELSEWHERE keyword may now be spelled as two words ‘ELSE WHERE’ in free source form.

• The intrinsic subroutine SYSTEM_CLOCK now accepts any kind of integer argument, and accepts real arguments for the COUNT_RATE= argument. Note that specifying a small integer kind here is likely to produce the wrong answer, as may single precision on systems with a fast clock.

• BOZ constants are now allowed as arguments to the intrinsic functions CMPLX, DBLE, INT and REAL.

• The intrinsic subroutine DATE_AND_TIME now accepts short CHARACTER arguments (this will lose information and so is not recommended).

4 Performance Enhancements

• Assignment of a structure constructor to a variable of derived type now avoids using a temporary when possible.

• The 2-argument form of the intrinsic function ASSOCIATED is faster in some cases.

• The intrinsic function ISHFTC is much faster for large SHIFT= values.

• The intrinsic function SHAPE is now faster in some cases.

• The intrinsic function SIZE with a DIM= argument is now faster in some cases.

• Allocation and deallocation of scalar pointers is now faster.

•Deallocation of allocatable arrays is sometimes faster if the −Oinline_dealloc option is used (this is the default for −O3 and −O4).

• The performance of the memory allocator has been improved, with a slight reduction in memory fragmentation.

• Reduced lock contention for multi-threaded programs with multiple threads each doing external input/output (to different units).

• Improved performance with the −thread_safe option for internal input/output.

• Compilation speed has been improved, especially for modules containing very large numbers of nested scoping units.
5 Other Major Enhancements

- Quad precision REAL is now supported on all platforms. On all platforms except Solaris with the Sun C compiler (product NPSOL52NA), this is “double double” precision; this provides approximately twice the precision of (64-bit) double precision, but has a smaller exponent range. It is not compatible with IEEE quad precision and is not as accurate, but is much faster.
- Nearly all runtime error messages now specify the location of the error in the message; in some cases this additional information is only available when compiled with −g or an appropriate checking (−C=) option.
- All non-standard intrinsic modules have been revised, making most module procedures generic, usable with the −double and −f77 options, improved error messages, et cetera.
- The intrinsic subroutines RANDOM NUMBER and RANDOM SEED now implement the “Mersenne Twister” random number generator. This has greatly improved “randomness” qualities (including sequence length) at approximately the same performance as the old generator.

6 Other Minor Enhancements

- Accept lines longer than 132 characters in free source form; there is no limit on line length, but an individual token is still limited to approximately 255*132 characters.
- Produce a warning when assigning a floating point constant expression to an integer variable if integer overflow would occur.
- Produce warnings when CASE values are not possible for the type of the selector.
- Produce a warning if a CASE range covers the entire set of possible values when there is also a CASE DEFAULT statement.
- Detect more invalid input/output specifier values at compile time.
- Detect −f77 option mismatch between modules and code using the module (this option changes the ABI so needs to be consistent).
- Detect (variable) stride value in a section subscript being zero at runtime.
- Integer overflow during exponentiation is detected at runtime if the −g option is used (32-bit and 64-bit integers only).
- The intrinsic function MOD with an integer argument now produces an informative error message instead of an arithmetic exception if the P= argument is zero.
- Additional information supplied for many compiler error messages, e.g. if a procedure reference has the wrong data type for an argument, it now says what type was expected and what type was supplied.
- Revised wording (previously inconsistent) in a number of compiler error messages.
- Split the compiler error message category ‘Warning’ into serious warnings (still labelled as ‘Warning’) and ‘Questionable’; the latter indicates code that is technically valid but which is frequently an accidental mistake.
- Produce ‘Questionable’ warning messages for:
  - using the intrinsic function CMPLX with double (or quad) precision arguments but without any KIND= argument;
− ordinary assignment of the result of a pointer function reference to a pointer variable;
− using a KIND= argument on an intrinsic function reference that specifies a small integer kind (i.e.
  1 or 2) when that is very likely to be too small to hold the result;
− apparently non-thread-safe code in a RECURSIVE procedure (only when the −thread_safe option
  is being used);
− passing a non-recursive procedure name as an actual argument from itself (this used to be an
  error, but is valid in Fortran 2003).

• Runtime error messages can be written to a specified file instead of ‘stderr’, by setting the environment
  variable NAGFORTRAN_RUNTIME_ERROR_FILE.
• Additional information supplied for some runtime error messages, e.g. saying which pointer is unde-
  fined (previously it only did this for simple names).
• New F90_STAT values STAT_MEMORY_LIMIT_EXCEEDED and STAT_PART_OF_LARGER_OBJECT.
• Use high-precision counter for SYSTEM_CLOCK on Windows.
• Increase maximum allocated memory size from 64GB to 1TB (64-bit systems only).
• The −thread_safe option now makes all user procedures thread-safe, not just RECURSIVE ones. Also, it
  now produces error messages if any non-recursive procedure updates a SAVEd variable or does anything
  else which is inherently not thread-safe.
• The ‘.ff’, ‘.ff90’ and ‘.ff95’ extensions are now recognised on all platforms as requesting fpp prepro-
  cessing (previously only on Windows and Mac OS/X).
• The ‘.f03’, ‘.F03’ (Unix only) and ‘.ff03’ extensions are now recognised (but not recommended or
  documented as there is no difference in source form between Fortran 90, 95 and 2003).
• The −indirect option has been added, which interprets the contents of a file as additional arguments
  to the compiler: this option may also be spelled as ‘@’.
• The −C=scale option has been removed since it is not compatible with Fortran 2003.
• The −C=recursion option is automatically disabled when the −thread_safe option is used.

7 New Fortran Standard

The extensions (described above) which follow the rules of the Fortran 2003 standard are listed below to-
gether with the appropriate section number for the reference book “Fortran 95/2003 Explained” by Metcalf,

• Unlimited polymorphic entities (16.3.1),
• Procedure pointers (15.6),
• Object-bound procedures (15.6.2),
• Allocatable scalars (17.5.1),
• Deferred character length (15.2),
• Intrinsic functions in initialisation expressions (17.10),
• Reallocating assignment (17.5.2),

• Unlimited polymorphic entities (16.3.1),
• Procedure pointers (15.6),
• Object-bound procedures (15.6.2),
• Allocatable scalars (17.5.1),
• Deferred character length (15.2),
• Intrinsic functions in initialisation expressions (17.10),
• Reallocating assignment (17.5.2),
• Recursive input/output (19.7),
• The associate construct (16.4),
• The intrinsic subroutine MOVE_ALLOC (17.5.3),
• KIND= arguments for intrinsic functions (18.10),
• CHARACTER arguments for intrinsic functions (18.10),
• Type-spec for an array constructor (17.9)
• Asynchronous input/output (19.3),
• Enhanced complex constants (18.12),
• Pointer lower bound setting (17.6),
• Renaming operators on the USE statement (17.8),
• The C_F_PROCPOINTER subroutine (14.3),
• Changes to SYSTEM_CLOCK (18.10),
• Allowing BOZ constants in COMPLEX, DBLE, INT and REAL (18.9).