Contents

Compatible $f77$ Features  2
Incompatibility Issues  5
Linking with $f77$-Compiled Routines  7
  Fortran 95 Intrinsics  8
Additional Notes About Migrating to the $f95$ Compiler  8
FORTRAN 77 Compatibility: Migrating to Fortran 95

This document is excerpted from the Forte Developer Fortran User's Guide.

The Fortran 95 compiler, $f95$, will compile most legacy FORTRAN 77 programs, including programs utilizing non-standard extensions previously compiled by the $f77$ compiler.

$f95$ will accept many of these FORTRAN 77 features directly. Others require compiling in FORTRAN 77 compatibility mode ($f95 -f77$).

This chapter describes the FORTRAN 77 features accepted by $f95$, and lists those $f77$ features that are incompatible with $f95$. For details on any of the non-standard FORTRAN 77 extensions that were accepted by the $f77$ compiler, see earlier releases of the FORTRAN 77 Language Reference manual on http://docs.sun.com.

See Chapter 4 of the Fortran User’s Guide for other extensions to the Fortran 95 language accepted by the $f95$ compiler.

$f95$ will compile standard-conforming FORTRAN 77 programs. To ensure continued portability, programs utilizing non-standard FORTRAN 77 features should migrate to standard-conforming Fortran 95. Compiling with the $-ansi$ option will flag all non-standard usages in your program.
Compatible $f77$ Features

$f95$ accepts the following non-standard features of the FORTRAN 77 compiler, $f77$, either directly or when compiling in $-f77$ compatibility mode:

- **Source Format**
  - Continuation lines can starting with ‘&’ in column 1. [$-f77=misc$]
  - The first line in an include file can be a continuation line. [$-f77=misc$]
  - Use $f77$ tab-format. [$-f77=tab$]

- **I/O:**
  - You can open a file with ACCESS=’APPEND’ in Fortran 95.
  - List-directed output uses formats similar to the $f77$ compiler. [$-f77=output$]
  - $f95$ allows BACKSPACE on a direct-access file, but not ENDFILE.
  - $f95$ allows implicit field-width specifications in format edit descriptors. For example, FORMAT(I) is allowed.
  - $f95$ will recognize $f77$ escape sequences (for example, \n \t \’) in output formats. [$-f77=backslash$.]
  - $f95$ recognizes FILEOPT= in OPEN statements.
  - $f95$ allows SCRATCH files to be opened or closed with STATUS=’KEEP’ [$-f77$]. When the program exits the scratch file is not deleted. SCRATCH files can also be opened with FILE=’name’ when compiling with $-f77$.
  - Direct I/O is permitted on internal files. [$-f77$]
  - $f95$ recognizes FORTRAN 77 format edit descriptors A, $, and SU. [$-f77$]
  - FORM=’PRINT’ can appear on OPEN statements. [$-f77$]
  - $f95$ recognizes the legacy FORTRAN input/output statements ACCEPT and TYPE.
  - Compile with $-f77=output$ to write FORTRAN 77 style NAMELIST output.
  - A READ with only ERR= specified (no IOSTAT= or END= branches) treats the ERR= branch as an END= when an EOF is detected. [$-f77$]
  - VMS Fortran NAME=’filename’ is accepted on OPEN statements. [$-f77$]
  - $f95$ accepts an extra comma after READ() or WRITE(). [$-f77$]
  - END= branch can appear on direct access READ with REC=. [$-f77=input$]
  - Allow format edit descriptor Ew.d.e and treat it as Ew.d.Ee. [$-f77$]
  - Character strings can be used in the FORMAT of an input statement. [$-f77=input$]
  - IOSTAT= specifier can appear in ENCODE/DECODE statements.
- List-directed I/O is allowed with `ENCODE/DECODE` statements.
- Asterisk (*) can be used to stand in for `STDIN` and `STDOUT` when used as a logical unit in an I/O statement.
- Arrays can appear in the `FMT=` specifier. [-f77=misc]
- `PRINT` statement accepts namelist group names. [-f77=output]
- The compiler accepts redundant commas in `FORMAT` statements.
- While performing `NAMELIST` input, entering a question mark (?) responds with the name of the namelist group being read. [-f77=input]

**Data Types, Declarations, and Usage:**

- In a program unit, the `IMPLICIT` statement may follow any other declarative statement in the unit.
- `f95` accepts the `IMPLICIT UNDEFINED` statement.
- `f95` accepts the `AUTOMATIC` statement, a FORTRAN 77 extension.
- `f95` accepts the `STATIC` statement and treats it like a `SAVE` statement.
- `f95` accepts VAX `STRUCTURE`, `UNION`, and `MAP` statements.
- `LOGICAL` and `INTEGER` variables can be used interchangeably. [-f77=logical]
- `INTEGER` variables can appear in conditional expressions, such as `DO WHILE`. [-f77=logical]
- Cray pointers can appear in calls to intrinsic functions.
- `f95` will accept data initializations using slashes on type declarations. For example: `REAL MHW/100.101/, ICOMX/32.223/`
- `f95` allows assigning Cray character pointers to non-pointer variables and to other Cray pointers that are not character pointers.
- `f95` allows the same Cray pointer to point to items of different type sizes (for example, `REAL*8` and `INTEGER*4`).
- `f95` accepts the `BYTE` data type.
- `f95` allows non-integers to be used as array subscripts. [-f77=subscript]
- `f95` allows relational operators `.EQ.` and `.NE.` to be used with logical operands. [-f77=logical]
- `f95` will accept the legacy `f77 VIRTUAL` statement, and treats it as a `DIMENSION` statement.
- Different data structures can be equivalenced in a manner that is compatible with the `f77` compiler. [-f77=misc]
- Like the `f77` compiler, `f95` allows many intrinsics to appear in initialization expressions on `PARAMETER` statements.
- f95 allows assignment of an integer value to CHARACTER*1 variables. [-f77=misc]
- BOZ constants can be used as exponents. [-f77=misc]
- An integer array of hollerith characters can be used as a format descriptor. [-f77]
- When compiling with -f77=misc, f95 will automatically promote a REAL constant to the appropriate kind (REAL*8 or REAL*16) in assignments, data, and parameter statements, in the manner of the f77 compiler. [-f77=misc]
- Equivalenced variables are allowed on an assigned GOTO. [-f77]
- Compiling with -f77=misc allows *kind before dimension declarations (for example, REAL X*8(21)). [-f77=misc]
- A character substring may appear as an implied-DO target in a DATA statement. [-f77=misc]
  For example: DATA (a(i:i), i=1,n) /n*'+'/  

Programs, Subroutines, Functions, Statements:  
- f95 does not require a PROGRAM statement to have a name.  
- Functions can be called by a CALL statement as if they were subroutines. [-f77]  
- Functions do not have to have their return value defined. [-f77]  
- An alternate return specifier (*label or &label) can appear in the actual parameter list and in different positions. [-f77=misc]  
- %VAL can be used with an argument of type COMPLEX. [-f77=misc]  
- Tab-formatting can extend source lines beyond column 72. [-f77]  
- f95 tab-formatting will not pad character strings to column 72 if they extend over a continuation line. [-f77]  
- A subroutine can call itself recursively without declaring itself with a RECURSIVE keyword. [-f77=misc]  
- Compiling with -f77=misc allows statement functions to be defined with arguments typed other than INTEGER or REAL, and actual arguments will be converted to the type defined by the statement function. [-f77=misc]  
- f95 treats a call to the function %LOC() as a call to LOC(). [-f77=misc]  

Miscellaneous:  
- The f95 normally does not issue progress messages to standard out. The f77 compiler did issue progress messages, displaying the names of the routines it was compiling. This convention is retained when compiling with the -f77 compatibility flag.  
- Programs compiled by the f77 compiler did not trap on arithmetic exceptions, and automatically called ieee_retrospective on exit to report on any exceptions that may have occurred during execution. Compiling with the -f77
flag mimics this behavior of the $f77$ compiler. By default, the $f95$ compiler 
traps on the first arithmetic exception encountered and does not call 
$\text{ieee\_retrospective}$. 

- The $f77$ compiler treated a REAL*4 constant as if it had REAL*8 precision in 
  contexts where double precision was needed. When compiling with the -$f77$
flag, the $f95$ compiler allows a REAL*4 constant to have REAL*8 precision 
when the constant is assigned to a REAL*8 variable. However, in arithmetic 
operations involving a REAL*4 constant and a REAL*8 operand, $f77$ treated 
the REAL*4 constant as if it were a REAL*8 constant. The $f95$ compiler does 
not do this, causing a possible difference in precision. 

A partial workaround for some applications is to compile with the -$r8$\text{const}$ 
flag to convert all REAL*4 constants to REAL*8. Note that this is not quite the 
$f77$ compiler’s behavior and could cause an interface problem if a REAL*4 
constant is used as an actual argument to a subprogram expecting REAL*4. It 
could also cause problems when reading back data written by programs that 
do unformatted writes with literal constants on the I/O list. 

For details on the syntax and semantics of non-standard language extensions, see the 
$\text{FORTRAN 77 Language Reference}$ on docs.sun.com. 

---

**Incompatibility Issues**

The following lists known incompatibility issues that arise when compiling and 
testing legacy $f77$ programs with this release of $f95$. These are due to either missing 
comparable features in $f95$, or differences in behavior. These items are non-standard 
extensions to Fortran 77 supported in $f77$ but not in $f95$. 

- **Source Format** 
  - $f95$ limits the number of continuation lines to 99. 

- **I/O:** 
  - Variable format expressions are not available in $f95$. 
  - $f95$ does not allow ENDFILE on a direct-access file. 
  - $f95$ does not recognize the '$n' form for specifying a record number in direct 
    access I/O: \texttt{READ (2 '13) X, Y, Z} 
  - $f95$ does not recognize the legacy $f77$ “R” format edit descriptor. 
  - $f95$ does not allow the DISP= specifier in a CLOSE statement. 
  - Bit constants are not allowed on a WRITE statement. 
  - Fortran 95 NAMELIST does not allow arrays and character strings with variable 
    lengths.
- Opening a direct access file with `RECL=1` cannot be used as a “stream” file. Use `FORMAT='STREAM'` instead.
- Fortran 95 reports illegal I/O specifiers as errors. `f77` gave only warnings.

**Data Types, Declarations, and Usage:**
- `f95` allows only 7 array subscripts; `f77` allowed 20.
- `f95` does not allow non-constants in `PARAMETER` statements.
- Integer values cannot be used in the initializer of a `CHARACTER` type declaration.
- Fortran 95 will not allow array elements in boundary expressions before the array is declared. For example:

```fortran
subroutine s(i1,i2)
  integer i1(i2(1):10)
  dimension i2(10)
  ...ERROR: "I2" has been used as a function, therefore it
  must not be declared with the explicit-shape DIMENSION
  attribute.
end
```

**Programs, Subroutines, Functions, Statements:**
- The maximum length for names is 31 characters.
- `f95` does not handle debugging comments (comment lines with "D" in column one). They are always treated as comments and there is no option `-vax=debug` to turn them into live statements.

**Command-line Options:**
- `f95` does not recognize the following `f77` compiler options:
  - `arg=local -dbl -oldstruct -i2 -i4 -r4 -r8 -vax`

**f77 Library Routines Not Supported by f95:**
- The POSIX library.
- The `IOINIT()` library routine.
- The tape I/O routines `topen, tclose, twrite, tread, trewin, tskipf, tstate`.
- `start_iostats` and `end_iostats` library routines.
- `f77_init()` function.
- `f95` does not allow the `IEEE_RETROSPECTIVE` subroutine to be bypassed by defining the user’s own routine with the same name.
Linking with f77-Compiled Routines

- To mix f77 and f95 object binaries, link with f95 compile with the -xlang=f77 option. Perform the link step with f95 even if the main program is an f77 program.

- Example: Compiling an f95 main program with an f77 object file.

```
<code>
demo% cat m.f95
CHARACTER*74 :: c = 'This is a test.'
   CALL echo1( c )
END
demo% f95 -xlang=f77 m.f95 sub77.o
demo% a.out
This is a test.
demo%
</code>
```

- The FORTRAN 77 library and intrinsics are available to f95 programs and are listed in the Fortran Library Reference Manual.

Example: f95 main calls a routine from the FORTRAN 77 library.

```
<code>
demo% cat tdtime.f95
   REAL e, dtime, t(2)
e = dtime( t )
DO i = 1, 100000
   as = as + cos(sqrt(float(i)))
END DO
e = dtime( t )
PRINT *, 'elapsed: ', e, ', user: ', t(1), ', sys: ', t(2)
END
demo% f95 tdtime.f95
demo% a.out
elapsed: 0.14 , user: 0.14 , sys: 0.0E+0
demo%
</code>
```

See dtime(3F).
Fortran 95 Intrinsics

The Fortran 95 standard supports intrinsic functions that FORTRAN 77 did not have. The full set of Fortran 95 intrinsics, including non-standard intrinsics, appears in the Fortran Library Reference manual.

If you use any of the intrinsic names listed in the Fortran Library Reference as a function name in your program, you must add an EXTERNAL statement for $f95$ to use your routine rather than the intrinsic one.

The Fortran Library Reference also lists all the intrinsics recognized by earlier releases of the $f77$ compiler. The $f95$ compiler recognizes these names as intrinsics as well.

Compiling with $-f77=intrinsics$ limits the compiler’s recognition of intrinsic functions to just those that were known to the $f77$ compiler, ignoring the Fortran 95 intrinsics.

Additional Notes About Migrating to the $f95$ Compiler

- The floatingpoint.h header file replaces f77_floatingpoint.h, and should be used in source programs as follows:
  
  ```
  #include "floatingpoint.h"
  ```

- Header file references of the form $f77$/filename should be changed to remove the $f77$/directory path.

- Some programs utilizing non-standard aliasing techniques (by overindexing arrays, or by overlapping Cray or Fortran pointers) may benefit by compiling with the appropriate $-xalias$ flag. This is discussed with examples in the chapter on porting “dusty deck” programs in the Fortran Programming Guide.