Chapter 11

Compiler Options

This chapter contains the following subsections:

- "OPTIONS Statement"
- "In−Line Options"
- "$INCLUDE Statement"

This chapter describes options that affect source programs both during compilation and at run time. Execute these options using

- **OPTIONS** statement—specified in the source code as the first statement of a program unit
- In−line options—individual statements embedded in the source code
- **$INCLUDE** statement—includes Fortran source statements from an external library into a program

This chapter discusses these options according to the mechanisms used to specify them. The command line options, which are parameters specified as part of the *f77* command when the compiler is invoked, are explained in the *Fortran 77 Programmer's Guide.*

OPTIONS Statement

The **OPTIONS** statement has the following syntax:

`OPTIONS` *option*`[` *option*`...]`

where *option* can be any of the following:

`/I4   /NOI4   /F77   /NOF77   /CHECK=BOUNDS   /CHECK=NOBOUNDS`
`/EXTEND_SOURCE   /NOEXTEND_SOURCE`

These options perform the same function as the like−named command line options. See Chapter 1 of the *Fortran 77 Programmer's Guide* for a description of these options. Specifying *option* overrides a command line option when they are the same. *option* must always be preceded by a slash (/).

Use the following rules when specifying an **OPTIONS** statement:

- The statement must be the first statement in a program unit and must precede the **PROGRAM**, **SUBROUTINE**, **FUNCTION**, and **BLOCKDATA** statements.
- *option* remains in effect only for the duration of the program unit in which it is defined.

In−Line Options

The syntax for in−line compiler options consists of a dollar sign ($) in column 1 of a source record, followed by the name of the compiler option in either uppercase or lowercase, with no intervening blanks or other separators.

When an in–line compiler option is encountered in a source file, that option is put into effect beginning with the source statement following the in–line compiler option. The sections that follow describe the in–line compiler options supported by the compiler.

The compiler does not support the following options, but, for compatibility with other compilers, it does recognize them:

ARGCHECK	NOTBINARY
BINARY	SEGMENT
CHAREQU	SYSTEM
NOARGCHECK	XREF

When it encounters one of these options, the compiler issues a warning message and treats it as a comment line.

$COL72 Option

The **$COL72** option instructs the compiler to process all subsequent Fortran source statements according to the fixed–format 72–column mode described under Source Program Lines in Chapter 1. The compiler command line option **-col72** has an identical effect on a global basis.

$COL120 Option

The **$COL120** option instructs the compiler to process all subsequent Fortran source statements according to the fixed–format 120–column mode. The compiler command line option **-col120** has an identical effect on a global basis.

$F66DO Option

The **$F66DO** option instructs the compiler to process all subsequent **DO** loops according to the rules of Fortran 66. This principally means that all **DO** loop bodies will be performed at least once regardless of the loop index parameters. The compiler command line option **-onetrip** has the identical effect on a global basis.

$INT2 Option

The **$INT2** option instructs the compiler to make **INTEGER*2** the default integer type and **LOGICAL*1** the default logical type. This convention stays in effect for the remainder of the program and involves any symbolic names that are assigned a data type either by implicit typing rules or by using **INTEGER** or **LOGICAL** declaration statements without a type length being specified. This option is similar to the **-i2** command line option except for the effect on the default logical type.

$LOG2 Option

The **$LOG2** option instructs the compiler to make **LOGICAL*2** instead of **LOGICAL*4** the default type for **LOGICAL**. This convention stays in effect for the remainder of the program and involves any symbolic names that are assigned a data type either by implicit typing rules or by using the **LOGICAL** declaration statement without a type length being specified.

$INCLUDE Statement

The **$INCLUDE** statement includes source lines from secondary files in the current primary source program. This feature is especially useful when two or more separately compiled source programs require an identical sequence of source statements (for example, data declaration statements).

The form of the **$INCLUDE** statement is

`$INCLUDE` *filename*

where *filename* is either an absolute or relative UNIX file name. If the filename is relative and no file exists by that name relative to the current working directory, an error is given and no attempt is made to search an alternative path. The material introduced into the source program by the **$INCLUDE** statement will follow the **$INCLUDE** statement, beginning on the next line. Nesting of **$INCLUDE** statements is permitted within the constraints of the operating system.