CF90™ Ready Reference for CRAY T3E Systems

SQ–3900 2.0

Cray Research, Inc.

Portions of this product may still be in development. The existence of those portions still in development is not a commitment of actual release or support by Cray Research, Inc. Cray Research, Inc. assumes no liability for any damages resulting from attempts to use any functionality or documentation not officially released and supported. If it is released, the final form and the time of official release and start of support is at the discretion of Cray Research, Inc.

The CF90 compiler includes United States software patents 5,247,696, 5,257,372, and 5,361,354.

The UNICOS operating system is derived from UNIX® System V. The UNICOS operating system is also based in part on the Fourth Berkeley Software Distribution (BSD) under license from The Regents of the University of California.

Record of Revision

The date of printing or software version number is indicated in the footer.

Version	*Description*
1.0	December 1993. Original printing. Supports CF90 compiler release 1.0.
1.1	June 1994. Supports CF90 compiler release 1.0 on Cray PVP systems. Includes minor documentation updates.
2.0	November 1995. Supports CF90 compiler release 2.0 on CRAY T3E systems. Implementation of the CF90 compiler on CRAY T3E systems is deferred.

Preface

This publication describes the CF90 2.0 compiler release running on CRAY T3E systems. Implementation of the CF90 compiler on CRAY T3E systems is deferred.

The following documents also describe the CF90 compiler:

- *CF90 Commands and Directives Reference Manual*, publication SR–3901
- *CF90 Fortran Language Reference Manual, Volume 1*, publication SR–3902
- *CF90 Fortran Language Reference Manual, Volume 2*, publication SR–3903
- *CF90 Fortran Language Reference Manual, Volume 3*, publication SR–3905

Conventions

The following conventions are used throughout this manual:

Convention	Meaning
`command`	This fixed-space font denotes literal items such as commands, files, routines, path names, signals, messages, and programming language structures.
variable	Italic typeface denotes variable entries and words or concepts being defined.
[ ]	Brackets surround optional portions of a syntax statement.
...	Ellipses indicate that a preceding command-line element can be repeated.
≡	Statement continuation symbol.
Cray PVP systems	This term includes CRAY T90, CRAY C90, CRAY J90, CRAY EL, CRAY Y-MP E, and CRAY Y-MP M90 systems running the UNICOS 8.0, or later , operating system.

In this publication, *Cray Research*, *CRI* , and *Cray* refer to Cray Research, Inc. and/or its products.

Ordering publications

The *User Publications Catalog*, publication CP–0099, lists all Cray Research hardware and software manuals that are available to customers. Cray Research customers who subscribe to the Cray Inform (CRInform) program can access this information on the CRInform system.

To order a manual, either call the Distribution Center in Mendota Heights, Minnesota, at +1–612–683–5907 or send a facsimile of your request to fax number +1–612–452–0141. Cray Research employees may choose to send electronic mail to `orderdsk` (UNIX system users).

Customers who subscribe to the CRInform program can order software release packages electronically by using the `Order Cray Software` option.

Reader comments

If you have comments about the technical accuracy, content, or organization of this manual, please tell us. You can contact us in any of the following ways:

- Send us electronic mail from a UNICOS or UNIX system, using the following UUCP address:

  `uunet!cray!publications`

- Send us electronic mail from any system connected to Internet, using the following Internet addresses:

  `publications@timbuk.cray.com`

- Contact your Cray Research representative and ask that a Software Problem Report (SPR) be filed. Use `PUBLICATIONS` for the group name, `PUBS` for the command, and `NO-LICENSE` for the release name.

- Call our Software Publications Group in Eagan, Minnesota, through the Customer Support Call Center, using either of the following numbers:

  1–800–950–2729 (toll free from the United States and Canada)

  +1–612–683–5600

- Send a facsimile of your comments to the attention of "Software Publications" in Eagan, Minnesota, at fax number +1–612–683–5599.

We value your comments and will respond to them promptly .

Contents

f90(1) command

The f90(1) command invokes the CF90 compiler. This command has several options, and some options are used more often than others. The most commonly used options are as follows:

```
f90 [-d disable ] [-D var [,var ]] [-e enable ]
  [-f source_form ] [-I incldir ] [-m msg_lvl ]
  [-O opt [, opt ...]] [-p module_site ] [-r list_opt ]
  [-U var [,var ]] [--] file .suffix [90]
  [file .suffix [90]] ...
```

The full syntax of the command is as follows:

```
f90 [-b bin_obj_file ] [-c] [-C cifopts ]
  [-d disable ] [-D var [,var ]] [-e enable ]
  [-f source_form ] [-F] [-g] [-G debug_lvl ]
  [-i intlen ] [-I incldir ] [-l lib ] [-L dir ]
  [-m msg_lvl ] [-M msgs ] [-N col] [-o out_file ]
  [-O opt [, opt ...]] [-p module_site ] [-r list_opt ]
  [-R runchk ] [-s size ] [-S source_file ] [-T]
  [-U var [,var ]] [-v] [-V] [-Wl"loader_opt "]
  [-Wr"cflist_opt "] [-x dirlist ] [-x npes ] [--]
  file .suffix [90] [file .suffix [90]] ...
```

Some options are not available on all platforms. If you specify an option that is not supported on your platform, a message is issued and compilation continues when possible.

-b *bin_obj_file*

The -b *bin_obj_file* option disables the load step and saves the binary object file version of your program in *bin_obj_file* .

The -b *bin_obj_file* specification is ignored if it appears on the command line with either the -e S or the -S *source_file* options.

The -c option is redundant when specified with -b *bin_obj_file* because -b *bin_obj_file* automatically disables the load step.

Only one input file is allowed when the `-b` option is specified. If you have more than one input file, you can use the `-c` option to disable the load step and save the binary files to their default file names. If you have only one input file and neither the `-b` nor the `-c` options are specified, the binary version of your program is not saved.

By default, the binary file is saved in *file*`.o`, where *file* is the name of the source file and `.o` is the suffix used.

`-c`

The `-c` option disables the load step and saves the binary object file version of your program in *file*`.o`, where *file* is the name of the source file and `.o` is the suffix used. Using this option in conjunction with other options can lead to the following possible outcomes:

- If `-b` *bin_obj_file* is specified in conjunction with `-c`, the binary file is placed in *bin_obj_file* .
- If `-c` is specified in conjunction with `-o` *out_file* and there is only one input file, the binary file is saved in *out_file* .
- If you specify the `-c` option and there is more than one input file, a *file*`.o` is created for each input file specified.

If only one input file is processed and neither the `-b` nor the `-c` options are specified, the binary version of your program is not saved after the load is completed.

`-C` *cifopts*

The `-C` *cifopts* option creates a compiler information file (CIF) and places it in *file*`.T`, where *file* is the name of the source file and `.T` is the suffix used. If both the `-r` and `-C` options are specified, `-r` overrides `-C`.

By default, the `f90` command does not create a CIF. The `-C` option is used to create a CIF, which is required by various tools that analyze CF90 programs, such as `cflint`, `cflist`, and `xbrowse`. The user interface to the CIF information is through the CIF library, `libcif`.

The most common specification is `-C a`, which writes all CIF information possible. For information on the *cifopts* available, see section 4 of the *Compiler Information File (CIF) Reference Manual*, publication SR–2401.

`-d` *disable* and `-e` *enable*

The `-d` *disable* and `-e` *enable* options disable or enable compiling options. To specify more than one compiling option, enter the options without separators between them; for example, `-e af`. Enter one or more of the following for *disable* or *enable* :

Table 1. Compiling options

disable *enable*	Action if enabled
`a` Disabled	Aborts compilation after encountering the first error.
`i` Disabled	Generates a run-time error when an uninitialized local variable is used in a floating-point operation or array subscript. This option causes allocated but uninitialized local storage to be set to an undefined value. When the `-e i` option is specified, variables are reset for each invocation of a subprogram, even in static storage. Therefore, a `SAVE` statement is needed to preserve the value of a variable between invocations. This option does not apply to variables in common blocks (or those equivalenced to such variables), variables in a `DATA` statement, dummy arguments, or saved variables. However, the loader offers this feature for saved variables that were not allocated in an `ALLOCATE` statement. If the compiler detects use of an uninitialized variable, it generates a message; this feature is unrelated to the `-e i` option and does not apply to arrays.
`n` Disabled	Generates messages to note all nonstandard usage.

Table 1. Compiling options (continued)

disable *enable*	Action if enabled
`p` Enabled	Enables double precision arithmetic. When disabled, variables declared on a `DOUBLE PRECISION` statement and constants specified with the `D` exponent are implicitly converted to default real type. This causes arithmetic operations and intrinsics involving these variables to have a default real type rather than a double-precision real type. Similarly, variables declared on a `DOUBLE COMPLEX` statement and complex constants specified with the `D` exponent are implicitly mapped to the complex type in which each part has a default real type. (The implementation of double precision arithmetic is deferred on CRAY T3E systems.)
`q` Enabled	Aborts compilation if 100 or more errors are generated.

`u` Disabled	Rounds floating-point division results upwards.
	
	When disabled, faster code sequences are generated for floating-point divides. This can result in slightly less accurate results because when results of floating-point divides are later assigned to integer variables, truncation errors can occur. For example, the following code fragment may result in the value of `i`, an integer, being truncated to an unexpected value:
	
	`x = 6.0`
	`r = x/3.0`
	`...`
	`i = r`
	
	On CRAY T3E systems, floating-point division may not produce correctly rounded results when disabled because of the faster divide sequence that is used. This may affect the results when converting to an integer value or when comparing results with those obtained from another system that supports IEEE floating-point arithmetic.

Table 1. Compiling options (continued)

disable *enable*	Action if enabled
	When enabled, additional instructions are generated that produce more accurate results. On systems that support IEEE floating-point arithmetic, a true IEEE divide sequence is generated (instead of possibly calculating a reciprocal and multiply sequence).
`v` Disabled	Allocates variables to static storage. These variables are treated as if they had appeared in a `SAVE` statement. Two types of variables are not allocated to static storage: variables allocated in an `ALLOCATE` statement and local variables in explicit recursive procedures. Variables in explicit recursive procedures consist of those in functions, in subroutines, and in internal procedures within functions and subroutines that have been declared with the `RECURSIVE` attribute. The `STACK` compiler directive overrides `-e v`.
`A` Disabled	Generates code necessary to use the MPP Apprentice tool.
`B` Enabled	Generates binary output. If disabled, inhibits all optimization and allows only syntactic and semantic checking.
`I` Disabled	Treats all variables as if an `IMPLICIT NONE` statement had been specified. Does not override any `IMPLICIT` statements or explicit type statements. All variables must be typed.
`P` Disabled	Performs source preprocessing on the *file*`.f[90]` or *file*`.F[90]` but does not compile. Generates *file*`.i`.
`R` Disabled	Compiles all functions and subroutines as if they contained a `RECURSIVE` keyword.

Table 1. Compiling options (continued)

disable *enable*	Action if enabled
S Disabled	Generates assembly language output and saves it in *file*.s. When the -e S option is specified on the command line with the -S *source_file* option, the -S *source_file* option overrides the -e S option. If -e S is specified with the -b *bin_obj_file* option, the -b *bin_obj_file* option is ignored.
Z Disabled	Performs source preprocessing and compilation on *file*.f[90] or *file*.F[90]. Generates *file*.i.

−D *var* [, *var*] ...

Defines variables used for conditional compilation. If the input file name ends in .F or .F90, GPP is invoked and it recognizes the −D and −U options on the f90(1) command line. If GPP is not invoked, these options are recognized directly by the CF90 compiler. For more information on GPP, see gpp(1).

The −U option undefines variables used for conditional compilation.

For more information on conditional compilation, see the *CF90 Fortran Language Reference Manual, Volume 3*, publication SR–3905.

−f *source_form*

The −f *source_form* option specifies whether the Fortran source file is written in fixed source form or free source form. For *source_form*, enter free or fixed. The *source_form* specified here overrides any source form implied by the source file suffix.

The default source form is fixed for input files that end with a .f or .F suffix. The default source form is free for input files that end with a .f90 or .F90 suffix.

If the file ends in .F or .F90, GPP is invoked. For more information on GPP, see gpp(1).

`-F`

Enables macro expansion throughout the entire source file. Typically, macro expansion occurs only on directive lines. If the source file suffix is neither `.F` nor `.F90`, this option is ignored.

`-g`

The `-g` option provides debugging support identical to specifying the `-G 0` option.

`-G` *debug_lvl*

The `-G` *debug_lvl* option generates a debug symbol table and establishes a debugging level. The debugging level determines the points at which breakpoints can be set. The frequency and position of breakpoints can curtail optimization partially or totally. At higher debugging levels, fewer breakpoints can be set, but optimization is increased. Enter one of the following for *debug_lvl* :

debug_lvl	Support
0	Default debugging support. Breakpoints can be set at each line. This level of debugging can be specified when `-O 0` is specified. Disables scalar optimization, tasking, and most vectorization; array syntax statements vectorize at this level. This level can also be obtained by specifying the `-g` option.
1	Performs block-by-block debugging, with the exception of innermost loops. You can place breakpoints at statement labels on executable statements and at the beginning and end of block constructs (such as `IF/THEN/ELSE` blocks, `DO/END DO` blocks, and at `SELECT CASE/END SELECT` blocks). This level of debugging can be specified when `-O 0` or `-O 1` is specified. Disables some scalar optimization and all loop nest restructuring.

`-i` *intlen*

The `-i` *intlen* option specifies 64-bit, 46-bit, or 32-bit integer arithmetic for default integers. Enter `64`, `46`, or `32` for *intlen* . The default *intlen* value for CRAY T3E systems is `64`.

Warning : Specifying `-i 46` is obsolescent. It causes the compiler to generate a warning message. This functionality will be removed in the CF90 3.0 release, and the new Cray PVP system default will be `-i 64`.

This option specifies 64-bit, 46-bit, or 32-bit integer arithmetic for operations involving multiplication, division, conversion to or from real, and comparisons. Specifying 32-bit or 46-bit integer arithmetic does not limit all representations to 32 or 46 bits, but it enables the compiler to generate faster code for some expressions. Specifying `64` generates a full 64-bit code sequence and performance is slower . To increase performance, you can specify `-O fastint` in conjunction with `-i 64`.

A value specified in an `INTEGER` compiler directive overrides this option.

If a module is compiled separately from a program unit that uses the module, they both must be compiled with the same *intlen* . If program units are compiled separately and linked, they must all be compiled with the same *intlen* .

`-I` *incldir*

The `-I` *incldir* option specifies a directory to be searched for files named in `INCLUDE` lines in the Fortran source file. You must specify an `-I` option for each directory you want searched. Directories can be specified in *incldir* as full path names or as path names relative to the working directory.

The following example causes the compiler to search for files included within `earth.f` in the directories `/usr/local/sun` and `../moon`:

```
f90 -I /usr/local/sun -I ../moon earth.f
```

If the `INCLUDE` line in the source file specifies an absolute name, that is, one that begins with a slash (`/`), that name is used, and no other directory is searched. If the `INCLUDE` uses a relative name, that is, one that does not begin with a slash (`/`), the compiler searches for the file in the directory of the source file containing the `INCLUDE` line. If this directory contains no file of that name, the compiler then searches the directories named by the `-I` options, as specified on the command line, from left to right.

`-l` *lib*

The `-l` *lib* option directs the loader to search for the specified object library file when loading an executable file.

If *lib* begins with a period (.) or a slash (/), it is assumed to be a full path name, and the loader uses it as is. Otherwise, the loader searches for a file named `lib`*lib*`.a` in each directory specified in the library search path.

`-L` *dir*

The `-L` *dir* option directs the loader to search for object library files in the specified directory before searching in the standard directories. The loader searches for library files specified by `-l` options in the directories specified by preceding `-L` options.

dir can take the form of a comma-separated list of directories. The loader searches for library files in directory *dir* before checking in the standard directories. The standard system directories are `/lib` and `/usr/lib`.

See `mppldr`(1) for more information on library searches.

`-m` *msg_lvl*

The `-m` *msg_lvl* option specifies the minimum compiler message levels to enable. The following list shows the integers to specify in order to enable each type of message and which messages are generated by default.

msg_lvl	Message types enabled
0	Error , warning, caution, note, and comment
1	Error , warning, caution, and note
2	Error , warning, and caution
3	Error and warning (default)
4	Error

Caution and warning messages denote, respectively, possible and probable user errors.

By default, messages are sent to the standard error file, `stderr`, and are displayed on your terminal. If the `-r` option is enabled, messages are also sent to the listing file.

To see more detailed explanations of messages, use the `explain` command. This command retrieves message explanations and displays them online. For example, to obtain documentation on message 500, enter the following command:

```
explain cf90-500
```

The default *msg_lvl* is `3`, which suppresses messages at the comment, note, and caution level.

`-M` *msgs*

The `-M` *msgs* option suppresses specific messages. For *msgs* , specify one or more integer numbers that correspond to the CF90 messages you want to suppress. If you want to specify more than one message number, enter a comma (but no spaces) between the message numbers. For example, to disable messages `cf90-100` and `cf90-200`, specify `-M 100,200`.

`-N` *col*

The `-N` *col* option specifies the line width for fixed-format source lines. For *col*, enter `72` to specify 72-column lines, enter `80` to specify 80-column lines, and enter `132` to specify 132-column lines.

By default, fixed-format source lines are 72 characters wide.

`-o` *out_file*

The `-o` *out_file* option overrides the default executable file name, `a.out`, with *out_file* . If this option is specified on the command line with the `-c` option, the binary object file is written to *out_file* and the load step is disabled. If this option is specified on the command line with the `-e S` or the `-S` options, the `-o` *out_file* option has no effect.

`-O` *opt* `[,` *opt* `]`...

You can use the `-O` *opt* option of the `f90`(1) command line to specify the following optimization features:

- General optimization levels. The `-O 0`, `-O 1`, `-O 2`, and `-O 3` options allow you to specify a general level and type of optimization. The range is from no optimization, at level `-O 0`, to aggressive optimization, at `-O 3`. Generally , as the level increases, so does execution performance. Each level implies a certain level of scalar optimization.

  With increased execution performance, however, you may notice increases in compile time and compile size. You may also notice slight differences in results at different optimization levels.

- Specific optimization levels. The specific *opt* values, such `scalar2`, `scalar3`, and so on, determine the extent of scalar optimization that the compiler will perform.

Note: You can use only one method of specifying scalar optimization. The *opt* values `0`, `1`, `2`, and `3` cannot be specified in conjunction with *opt* values `scalar0`, `scalar1`, `scalar2`, or `scalar3`.

- Specific optimization characteristics. The `-O` option accepts arguments to control bottom loading of loops, pattern matching, zero incrementing, and several other optimization features. Some of these features can also be controlled through compiler directives.

CRAY T3E system defaults: `2`, `scalar2`, `noaggress`, `nobl`, `jump`, `nomsgs`, `nonegmsgs`, `nooverindex`, `pattern`, `recurrence`, and `unroll0`.

The following table summarizes the optimization features enabled when different *opt* levels are specified. The optimization specifications available differ depending on your platform. If your command line includes an option that is not supported on your platform, the compiler issues a message and compilation continues.

Table 2. `-O` *opt* summary

opt	Description
0	No optimization. Low compile time, small compile size, and no global scalar optimization.
1	Conservative optimization. Moderate compile time and size, global scalar optimization, no loop nest restructuring. Because of operator reassociation, results can differ from results obtained from `-O 0`. No optimizations performed that could create false exceptions.

Table 2. `-O` *opt* summary (continued)

opt	Description
2	Moderate optimization. Moderate compile time and size, global scalar optimizations, pattern matching, and loop nest restructuring. The system does not perform optimizations that might create false exceptions. Results can differ from results obtained when `-O 1` is specified. Default general optimization level.
3	Aggressive optimization. Larger compile size, longer compile time, global scalar optimization, possible loop nest restructuring, pattern matching, and Autotasking. Optimizations performed might create false exceptions in rare instances. Results can differ from results obtained when `-O 1` is specified.
`scalar0`	No scalar optimization. Low compile time and size.
`scalar1`	Conservative scalar optimization. Moderate compile time and size. Results can differ from results obtained when `scalar0` is in effect because of operator reassociation. No optimizations are performed that could create false exceptions.

Table 2. `-O` *opt* summary
(continued)

opt	Description
`scalar2`	Moderate scalar optimization. Moderate compile time and size. Results can differ from results obtained when `scalar1` is in effect because of possible loop nest restructuring. Generally, no optimizations are done that could create false exceptions. Default scalar optimization level.
`scalar3`	Aggressive scalar optimization. Potentially greater compile time and size. Results can differ from results obtained when `scalar1` is specified because of loop nest restructuring. Optimizations performed can create false exceptions in rare instances. Analysis that determines whether a variable is used before it is defined is enabled. `scalar3` is never enabled automatically , even when `-O 3` is specified; it must be requested specifically on the command line.
`aggress`, `noaggress`	`aggress` raises the limits for internal tables, which increases opportunities for optimization. The default is `noaggress`.
`bl`, `nobl`	`bl` enables full bottom loading of scalar operands in loops. The default is `nobl`.
`jump`, `nojump`	`jump` generates jumps instead of branches to external functions. The default is `jump`.

Table 2. -O *opt* summary
(continued)

opt	Description
msgs, nomsgs	msgs causes the compiler to write optimization messages to stderr. The default is nomsgs.
negmsgs, nonegmsgs	negmsgs writes negative optimization messages to stderr. The default is nonegmsgs.
overindex, nooverindex	nooverindex declares that there are no array subscripts that index a dimension of an array that are outside the declared bounds of that dimension. The default is nooverindex.
pattern, nopattern	pattern enables pattern matching for library substitution. The default is pattern.
recurrence, norecurrence	recurrence rewrites multiplication operations to be a series of addition operations. The default is recurrence.
split0, split1, split2	Specifies various levels of loop splitting. The default is split1.

Table 2. `-O` *opt* summary (continued)

opt	Description
`unroll0`, `unroll1`, `unroll2`	Specifies various levels of loop unrolling. `unroll0` disables loop unrolling and directs the compiler to ignore `UNROLL` compiler directives. `unroll1` causes the compiler to unroll only loops that are preceded by an `UNROLL` compiler directive. `unroll2` causes the compiler to evaluate all loops in the compilation as candidates for unrolling except those that are preceded by a `NOUNROLL` compiler directive. The default is `unroll0`.
`zeroinc`, `nozeroinc`	`zeroinc` causes the compiler to assume that increment variables (CIVs) can be incremented by zero. The default is `nozeroinc`, which means that you must prevent zero incrementing.

Note : If your command line includes an option that is not supported on your platform, the compiler issues a message and compilation continues.

`-p` *module_site*

The `-p` *module_site* option allows you to specify files or directories that contain modules.

The *module_site* specifies the name of a binary file or directory to search for modules. The *module_site* specified can be an archive file, build file (`bld` file), or binary file.

When searching files, the compiler searches files suffixed with `.o` (*file*`.o`) or library files suffixed with `.a` (*lib*`.a`) containing one or more modules. When searching directories, the compiler searches files in the named directory that are suffixed with `.o` or `.a`.

The module files are not interchangeable. For example, it is not possible to create a module with a CF90 compiler based on a SPARC system and read the file on a Cray PVP system.

File name substitution (such as *.o) is not allowed. If the path name begins with a slash (/), the name is assumed to be an absolute path name. Otherwise, it is assumed to be a path name relative to the working directory. If you need to specify multiple binary files, library files, or directories, you must specify a -p option for each *module_site* . There is no limit on the number of -p options that you can specify. The compiler searches the binary file, library files, and directories in the order specified.

Example 1: Consider the following command line:

```
f90 -p steve.o -p mike.o joe.f
```

Assume that steve.o contains a module called Rock and mike.o contains a module called Stone. A reference to use Rock in joe.f causes the compiler to use Rock from steve.o. A reference to Stone in joe.f causes the compiler to use Stone from mike.o.

Example 2: The following example specifies binary file murphy.o and library file molly.a:

```
f90 -p murphy.o -p molly.a prog.f
```

-r *list_opt*

The -r *list_opt* option produces a listing file. If the -C option is specified along with the -r option, the -C option is overridden and a warning message is generated. If both the -r option and the -Wr options are specified, the listing options specified by all the -r options are placed on the cflist command line before those specified by the -Wr option.

By default, if only one input file is specified on the f90(1) command line, the listing is placed in *file*.lst. If more than one input file is specified on the command line, the listing is placed in file cflist.out.

The -r option allows you to generate a listing directly from the f90 command. The *list_opt* argument produces listings with commonly needed information. If you need a customized report, you can use cflist directly.

The -r option accepts only one argument unless you are specifying the T argument; T can be specified in conjunction with one other argument.

The *list_opt* values are as follows:

list_opt	Listing type
0	Listing includes the standard `cflist` listing information. Includes global reports when you have more than one compilation unit.
1	Listing includes the source listing, loop marks, and parallel marks. Regardless of the number of compilation units, no global reports are included. This is a minimal listing.
2	Listing includes the source listing, loop marks, parallel marks, considerata, and compiler messages. Regardless of the number of compilation units, no global reports are included. Produces a short, standard listing.
3	Listing includes the source listing, loop marks, parallel marks, an argument report, a parallelism report, identifier cross-references, considerata, and compiler messages. Regardless of the number of compilation units, no global reports are included.
4	Listing includes all subprogram items that can be listed with the default `cflist` options. Regardless of the number of compilation units, no global reports are included.
5	Listing includes all subprogram items that can be listed with the default `cflist` options. If more than one compilation unit is present, the Global Considerata report is included.
6	Listing includes all items that can appear in a listing. All `cflist` options are enabled. This option produces the maximum amount of listing information.
`c`	Listing includes all common blocks and all members of each common block.
`g`	Saves the generated binary code and its assembly language equivalent to *file*`.L`. Unlike most other `-r` arguments, specifying `-r g` does not create a CIF.

list_opt	Listing type
l	Invokes cflint(1) with default cflint options. You cannot pass option information directly to cflint. If you require a particular cflint analysis, you must call it separately . If more than one source file is listed on the command line, the output is written to file cflint.out.
m	Produces a source listing with loopmark information. Loopmark information will not be displayed if the -d B option has been specified.
s	Lists source code. Messages are interspersed with the source lines. Produces 80-column output by default.
x	Generates a cross-reference listing. Produces 80-column output by default.
w	Produces 132-column output, which, when specified in conjunction with -r s or -r x, overrides the 80-column output that those options produce by default.
T	Retains *file*.T after processing rather than deleting it. This option may be specified in addition to any of the other options. For more information on *file*.T, see the -C option.

-R *runchk*

The -R *runchk* option lets you specify any of a group of run-time checks for your program. The run-time checks available are as follows:

runchk	Checking performed
a	Compares the number and types of arguments passed to a procedure with the number expected.
b	Enables checking of array bounds. If a problem is detected at run time, a message is issued but execution continues. The BOUNDS and NOBOUNDS compiler directives override this option.

runchk	Checking performed
`c`	Enables conformance checking of array operands in array expressions. Even without the `-R` option, such checking is performed during compilation when the dimensions of array operands can be determined.
`s`	Enables checking of character substring bounds. This option behaves similarly to option `-Ra` or `-Rb`.
`C`	Passes a descriptor for the actual arguments as an extra argument to the called routine and sets a flag to signal the called routine that this descriptor is included.
`E`	Creates a descriptor for the dummy arguments at each entry point and tests the flag from the caller to see if argument checking should be performed. If the flag is set, the argument checking is done.

If argument checking is to be done for a particular call, the calling routine must have been compiled with either `-R a` or `-R C` and the called routine must have been compiled with either `-R a` or `-R E`. `-R a` is equivalent to `-R CE`. The separation of `-R a` into `-R C` and `-R E` allows some control over which calls are checked.

Libraries can be compiled with `-R E`. If the program that is calling the libraries is compiled with either `-R a` or `-R C`, library calls are checked. If the calling routines are not compiled with `-R a` or `-R C`, no checking occurs.

Slight overhead is added to each entry sequence compiled with `-RE` or `-Ra` and to each call site compiled with `-RC` or `-Ra`. If a call site passes the extra information to an entry that is compiled to perform checking, the checking itself costs a few thousand clock periods per call. This cost depends on the number of arguments at the call.

Some nonstandard code behaves differently when argument checking is used. Different behavior can include run-time aborts or changed results. The following example illustrates this problem:

```
CALL SUB1(10,15)
CALL SUB1(10)
END

SUBROUTINE SUB1(I,K)
PRINT *,I,K
END
```

Without argument checking, if the two calls in this example share the same stack space for arguments, subroutine `SUB1` prints the values 10 and 15 for both calls. However, with argument checking enabled, an extra argument is added to the argument list, overwriting any previous information that was there. In this case, the second call to `SUB1` prints 10, followed by an incorrect value.

If full argument checking is enabled by `-Ra`, a message reporting the mismatch in the number of arguments is issued. This problem occurs only with nonstandard code in which the numbers of actual and dummy arguments do not match.

`-s` *size*

The `-s` *size* option allows you to modify the sizes of variables, literal constants, and intrinsic function results declared as type real, integer, logical, complex, double complex, or double precision. For *size*, enter one of the following:

size	Action
`default32`	Adjusts the default sizes as follows: real, integer , and logical are set to 32 bits; complex and double precision are set to 64 bits; double complex is set to 128 bits.
`i`	Relaxes the strict data typing rules of Fortran 90 when more than one kind exists for a type. Specifying `-s i` causes data types to map to only the standard intrinsic FORTRAN 77 types and provides compatibility with the Cray Research CF77 compiling system.

If you specify `-s` *size* in conjunction with data declarations that include size in your source code, the default size is not used. For example, if you specify `REAL(KIND=`*n*`) R` or `COMPLEX(KIND=`*n*`) Z`, the values specified in *n* are used.

`-S` *source_file*

The `-S` *source_file* option specifies the assembly language output file name. When `-S` *source_file* is specified on the command line with either the `-e S` or `-b` *bin_obj_file* options, the `-e S` and `-b` *bin_obj_file* options are overridden.

`-T`

The `-T` option disables the compiler but displays all options currently in effect. The CF90 compiler generates information identical to that generated when the `-v` option is specified on the command line; when `-T` is specified, however , no processing is performed. When this option is specified, output is written to the standard error file (`stderr`).

`-U` *var* `[,` *var*] ...

Undefines variables used for conditional compilation. If the input file name ends in `.F` or `.F90`, GPP is invoked and it recognizes the `-D` and `-U` options on the `f90`(1) command line. If GPP is not invoked, these options are recognized directly by the CF90 compiler. For more information on GPP, see `gpp`(1).

The `-D` option undefines variables used for conditional compilation.

For more information on conditional compilation, see the *CF90 Fortran Language Reference Manual, Volume 3*, publication SR–3905.

`-v`

The `-v` option sends compilation information to the standard error file (`stderr`). The information generated indicates the compilation phases, as they occur, and all options and arguments being passed to each processing phase.

`-V`

The `-V` option directs each compilation phase to send a message containing version information to the standard error file (`stderr`). Unlike all other command-line options, you can specify this option without specifying an input file name; that is, specifying `f90 -V` is valid.

`-Wl"`*loader_opt* `"`

The `-Wl"`*loader_opt* `"` option passes *loader_opt* directly to the loader. For example, specifying `-Wl"-D MAP=FULL"` passes the argument `MAP=FULL` directly to the loader's `-D` option. For more information on loader options, see `segldr`(1), `mppldr`(1), or `ld`(1).

−Wr"*cflist_opt* "

The −Wr"*cflist_opt* " option passes *cflist_opt* directly to cflist(1). For example, specifying −Wr"−o cfile.o" passes the argument cfile.o directly to the cflist −o option; this directs cflist to override the default output listing and put the output file in cfile.o. If you specify the −Wr option, you must specify the −r option. For more information on the cflist options, see cflist(1).

−x *dirlist*

The −x *dirlist* option disables specified directives or specified classes of directives. For *dirlist* , enter one of the following:

dirlist	Directives disabled
all	All compiler directives and Autotasking directives.
dir	All compiler directives.
directive	One or more compiler directives or Autotasking directives. If specifying more than one, separate them with commas.
mic	All Autotasking directives.

−x *npes*

The −x *npes* option specifies the number of processor elements (PEs) in the partition.

− −

The −− symbol signifies the end of options. After this symbol, you can specify files to be processed. This symbol is optional. It may be useful if your input files begin with one or more dash (−) characters.

file .*suffix* [90] [*file* .*suffix* [90]] ...

This option names the file or files to be processed, where *suffix* is either an uppercase F or a lowercase f. The file suffixes indicate the content of each file and determine whether the compiler, assembler, or loader will be invoked.

If the file ends in .F or .F90, GPP is invoked. For more information on GPP, see gpp(1).

Files containing uncompiled Fortran code must be in one of the following forms:

- *file* `.f` or *file* `.F`, which indicates that this is a fixed source form file

- *file* `.f90` or *file* `.F90`, which indicates that this is a free source form file

The source form specified on the `-f` option overrides the source form implied by the file suffixes.

By default, several files are created during processing. The CF90 compiler adds a suffix to the *file* portion of the file name and places the files it creates into your working directory.

Assembly language output is sent to *file*`.s`. File names ending with `.s` are assembled, and the assembled code is written to the corresponding *file*`.o`.

The compiled code is written to *file*`.o` in the current directory.

You can specify precompiled `.o` files as input files. Input file names ending with `.o` are passed to the loader. The following information may be useful to you if you are using precompiled files:

If your input files include precompiled CF90, CFT77, Cray Standard C, or Cray C++ files, they should be specified as files suffixed with `.o` on the command line. Files suffixed with `.o`, including any `.o` files written by the CF90 compiler, are passed to the loader in the order in which they appear on the `f90` command line. If, however, loading has been disabled by specifying options `-c` or `-b` *bin_obj_file*, no files are passed to the loader.

The loader produces an executable file; by default, `a.out`. See the `-o` *out_file* option for information on specifying a different executable file. If only one source file is specified on the command line, the `.o` file is created and deleted. To retain the `.o` file, use the `-c` option to disable the loader.

The following is a file summary:

File	Type
`a.out`	Executable output file.
file`.a`	Library files to be searched for external references.
file`.f` or *file*`.F`	Input Fortran source file in fixed source form. If file ends in `.F` or `.F90`, GPP is invoked. For more information on GPP, see `gpp`(1).
file`.f90` or *file*`.F90`	Input Fortran source file in free source form. If file ends in `.F` or `.F90`, GPP is invoked. For more information on GPP, see `gpp`(1).
file`.lst`	Listing file.
file`.o`	Relocatable object file.

File	Type
file `.s`	Assembly language file.
file `.L`	File containing binary code and generated assembly language output.
file `.T`	CIF output file.
`cflint.out`	`cflint`(1) output file created when the `-r l` option is specified and there are multiple source files.
`cflist.out`	`cflist`(1) output file created when the `-r` option is specified to invoke `cflist`(1) and there are multiple source files.

The loader allows other file types. See the `segldr`(1), `mppldr`(1), and `ld`(1) man pages for more information on these files.

Environment Variables

Environment variables are predefined shell variables, taken from the execution environment, that determine some of your shell characteristics. Several environment variables pertain to the CF90 compiler. The CF90 compiler recognizes general and multiprocessing environment variables.

The environment variables apply to all compilations in a session. The following example shows how to set an environment variable:

With the standard shell, enter:

```
TARGET=cpu
export TARGET
```

With the C shell, enter:

```
setenv TARGET cpu
```

The following subsections describe the environment variables recognized by the CF90 compiler.

Many of the environment variables described in this section refer to the default system locations of programming environment components. If the CF90 programming environment has been installed in a nondefault location, see your system support staff for path information.

`CAL`

Path name of the assembler. The default is `/opt/ctl/bin/as`.

`CAL_M`

Path name of the MPP assembler. The default is `/opt/ctl/bin/cam`.

`CFLIST`

Path name of `cflist`(1) and `cflint`(1). The default is `/opt/ctl/craytools/craytools/bin/cflist`.

`CRI_F90_OPTIONS`

Specifies additional options to be attached to the command line. These options are added following the options specified directly on the command line. File names cannot appear. This file is inserted at the rightmost portion of the command line before the input files and binary files are listed.

This allows you to set the environment variable once and have the specified set of options used in all compilations. This is especially useful for adding options to compilations done with build tools.

For example, assume that this variable was set as follows:

```
setenv CRI_F90_OPTIONS -G0
```

With the variable set, a command-line specification of `f90 -c t.f` is equivalent to a command-line specification of `f90 -c t.f -G0`.

`LISTIO_PRECISION`

Controls the number of digits of precision printed by list-directed output. The `LISTIO_PRECISION` environment variable can be set to `FULL` or `PRECISION`.

- `FULL` prints full precision (default).
- `PRECISION` prints *x* or *x*+1 decimal digits, where *x* is value of the `PRECISION` intrinsic function for a given real value. This is a smaller number of digits, which usually ensures that the last decimal digit is accurate to within 1 unit. This number of digits is usually insufficient to assure that subsequent input will restore a bit-identical floating-point value.

`MPP_SEGDIR`

`mppldr`(1) directives. This variable contains one or more strings separated by semicolons. Each string may be either a `mppldr`(1) directive or the name of a file containing `mppldr`(1) directives.

`NLSPATH`

Message system variable. The `NLSPATH` environment variable affects compiler interactions with the message system. For more information on this environment variable, see the `catopen`(3) man page.

`NPROC`

Maximum number of processes to be run in the background. Compilations can be multiprocessed by setting the `NPROC` environment variable to a value greater than 1. You can set `NPROC` to any value, but large values can overload the system. For debugging purposes, `NPROC` should be set to 1. By default, `NPROC` is 1.

`SEGLDR_M`

Path name of the loader. The default is `/opt/ctl/bin/mppldr`.

`TARGET`

Cross-compiling environment variable. *Cross-compiling* is compiling a program on one system to execute on another.

The `TARGET` environment variable allows you to cross-compile from most Cray PVP systems to other Cray Research systems. Cross-compiling from a CRAY T90 system that supports IEEE floating-point arithmetic to other Cray PVP systems is not possible. Cross compiling from a CRAY T3E system to a Cray PVP system is not possible.

The target system is the type of machine upon which the code will be executed. The `TARGET` environment variable recognizes the following values:

Value	Generates code for
`cray-t3e`	CRAY T3E systems
`cray-ts`	CRAY T90 systems without support for IEEE floating-point arithmetic
`cray-ts, ieee`	CRAY T90 systems with support for IEEE floating-point arithmetic
`cray-c90`	CRAY C90 systems
`cray-ymp`	CRAY Y-MP E systems and CRAY Y-MP M90 systems
`cray-el`	CRAY EL systems
`cray-j90`	CRAY J90 systems

See the `target`(1) man page for more information on setting this environment variable.

It is recommended that you load your binary program on the target system. If you load it on the host system, issues over differences between the libraries needed on the target and host systems may arise. The `TARGET` settings do not automatically cause the correct library to be selected. In many cases, host and target libraries will be compatible, but there may be incompatibilities between host and target systems.

The need for a particular set of libraries, as well as the correct specification to identify the set, must be determined for your particular site. If you need to specify a set of libraries contained in a directory, use the `f90 -L` option, as follows:

```
f90 -L /lib/ylib pgm.f    # For a library directory
```

The following example shows how to compile a program on a CRAY EL system and execute it on a CRAY C90 system.

On the CRAY EL system:

1. Set the `TARGET` environment variable to `cray-c90` using the procedure described in this subsection.

2. Compile your program using a command line that includes the `-c` option, as follows:

   ```
   f90 -c bigfile.f
   ```

   The `-c` option disables the load step and returns `bigfile.o`.

On the CRAY C90 system:

1. Move file `bigfile.o` from the CRAY EL system to the CRAY C90 system.

2. Use the `f90` command to link `bigfile.o` with the correct libraries:

   ```
   f90 bigfile.o
   ```

The `TARGET` environment variable applies until the end of an interactive session, so it affects other commands that use this environment variable to modify different CPU types. You must cancel these settings if you do not want to use them. You can prevent unexpected effects by setting the environment variable in a script or in a `make` file; the setting is effective only within that file.

`TMPDIR`

Directory to contain temporary files. By default, the CF90 compiler creates temporary files in `/var/tmp`. You can specify a different location by setting `TMPDIR` to your chosen directory. If `TMPDIR` is not a valid directory, the CF90 compiler uses `/var/tmp`.

Compiler Directives

Compiler directives are lines inserted into source code that specify actions to be performed by the compiler. They are not Fortran 90 statements. A *compiler directive line* begins with the characters `CDIR$` or `!DIR$`. How you specify compiler directives depends on the source form you are using.

External naming directive

`!DIR$ NAME` (*fortran_name* `="`*external_name* `"`
`[,` *fortran_name* `="`*external_name* `"`] ...)

Allows you to specify a case-sensitive external name, or a name that contains characters outside of the Fortran character set, in a Fortran program. *fortran_name* is the name used for the object throughout the Fortran program. *external_name* is the external form of the name. Rules for Fortran naming do not apply to the *external_name* string; any character sequence is valid. You can use this directive, for example, when writing calls to C routines.

Scalar optimization directives

`!DIR$ BL`,
`!DIR$ NOBL`

`NOBL` causes the use of safe bottom loading within loops. `BL` causes the compiler to use standard bottom loading. Safe bottom loading involves a small cost in performance but removes a possible cause of run-time errors (indicated by an `Operand range error` message).

`!DIR SPLIT`,
`!DIR NOSPLIT`

`SPLIT` asserts that the subsequent `DO` loop can be split safely. The loop will also be stripmined. `NOSPLIT` inhibits loop splitting and overrides a `-O split2` command line specification.

`!DIR$ SUPPRESS` [*var* [, *var*] ...]

Suppresses scalar optimization for all variables or only for those specified at the point where the directive appears. This often prevents or adversely affects vectorization of any loop that contains `SUPPRESS`. *var* is a variable that is to be stored to memory. If no variables are listed, all variables in the program unit are stored.

Local use of compiler features directives

`!DIR$ BOUNDS [`*arnm* `[,`*arnm* `]...]`,
`!DIR$ NOBOUNDS [`*arnm* `[,`*arnm* `]...]`

Checks most array references at both compile time and run time to ensure that each subscript is within the array's declared size. *arnm* is the name of an array. The name cannot be a subobject of a derived type. When no array name is specified, the directive applies to all arrays.

`!DIR$ FREE`,
`!DIR$ FIXED`

Specifies whether the source code in the program unit is written in free source form or fixed source form.

`!DIR$ INTEGER=`*n*

Specifies the integer length to be used in multiplication and division operations within one program unit for default integer types. It does not apply to subsequent program units. `INTEGER` is allowed only in external program units; it cannot be specified in internal or module procedures. This compiler directive must immediately follow the program unit header statement (`PROGRAM`, `FUNCTION`, `SUBROUTINE`, `MODULE`, or `BLOCKDATA`). `INTEGER` applies at both compile time and run time. For *n*, specify `32`, `46`, or `64` integer length. **Note**: The `INTEGER=46` specification is valid only on Cray PVP systems, and it is obsolescent. If you specify `INTEGER=46`, a warning message is generated. This value for *n* will be removed in the CF90 3.0 release.

Storage directives

`!DIR$ CACHE_ALIGN` *var* `[,` *var* `] ...`

Aligns each specified variable on a cache line boundary. This is useful for frequently referenced variables. A *cache* is storage that can be accessed more quickly than conventional memory. A *cache line* is a division within a cache. `CACHE_ALIGN` can be used with message-passing programming. *var* is the list of common block names or PE-private variable names, separated by commas. If specifying a common block name, the names of the items in the common block need not be specified.

`!DIR$ COMMON` *b*[,*b*]...

Overrides the `-a taskcommon` command line specification for a given procedure. This information is not inherited from the parent scoping unit. All common blocks remain as single-copy common blocks when this compiler directive appears in a procedure. *b* is a common block name. The name must also appear as a common block name in a `COMMON` statement.

`!DIR$ STACK`

Causes the default storage allocation to be to the stack in the program unit that contains the directive.

`!DIR$ SYMMETRIC` [*var* [, *var* ...]]

Declares that a PE-private stack variable has the same local address on all PEs. This is useful for global addressing using the SHMEM library routines. For more information on the SHMEM library routines, see `intro_shmem(3)`. *var* is a variable that is PE private. Variables that are specified are allocated at the same local address on all PEs. There is an implicit barrier before the first executable statement in the routine.

Miscellaneous directives

`!DIR$ RECURRENCE`,
`!DIR$ NORECURRENCE`

Converts floating-point operations that involve multiplication or exponentiation by an induction variable to be a series of additions or multiplications of a value. This produces faster code but can create numeric differences. `RECURRENCE` rewrites multiplication operations. `NORECURRENCE` stops the compiler from rewriting multiplication operations for loops in which the order of evaluation is numerically significant.

`!DIR$ UNROLL` [*n*],
`!DIR$ NOUNROLL`

Specifies the total number of loop body copies to be generated. *n* must be a positive integer. If you specify a value for *n*, the compiler does not attempt to determine the number of copies to generate based on the number of inner loops in the loop nest. `UNROLL` should be placed immediately before the `DO` statement of the loop that should be unrolled. `NOUNROLL` inhibits loop unrolling and overrides a `-o unroll2` command line specification.

Summary of Forms

For a summary of the Fortran 90 statement forms, please consult the Cray PVP portion of this ready reference. The statements are common to both platforms.

Intrinsic Procedures

For a summary of the CF90 intrinsic procedures, please consult the Cray PVP portion of this ready reference. Most intrinsics are common across all platforms, but platform-specific information is included in the Cray PVP section on intrinsic procedures.