FUJITSU SEMICONDUCTOR
CONTROLLER MANUAL

CM81-00206-7E

FR FAMILY
SOFTUNE™ C/C++ COMPILER MANUAL
for V6

FUJITSU

FR FAMILY
SOFTUNE™ C/C++ COMPILER
MANUAL

for V6

FUJITSU SEMICONDUCTOR LIMITED

PREFACE

■ Objective of This Manual and Target Readers

This manual describes the Softune C/C++ compiler (hereinafter referred to as the C/C++ compiler) usage procedures and libraries.

This manual is prepared for persons who use the above-mentioned compiler and create and develop application programs in C and C++ language. Read this manual thoroughly before starting.

This manual is to be read by persons who have a basic knowledge of each MCU (Micro Controller Unit).

The compiler described in this manual conforms about C language to the American National Standard for Information Systems Programming Language C, X3.159-1989, which is abbreviated ANSI standard in this manual. Part of "ISO/IEC 14882:1998 Programming languages -- C++" is used to explain C++.

FR, the abbreviation of FUJITSU RISC controller, is a line of products of FUJITSU SEMICONDUCTOR Limited.

■ Trademarks

SOFTUNE is a trademark of Fujitsu Semiconductor Limited.

Windows is a registered trademark of Microsoft Corporation in the USA and/or other countries.

UNIX is a registered trademark that X/Open Co., Ltd. has licensed in the United States and other countries.

The company names and brand names herein are the trademarks or registered trademarks of their respective owners.

■ Structure of This Manual

This manual consists of 10 chapters and an Appendix:

CHAPTER 1 SOFTUNE C/C++ COMPILER

This chapter outlines the C/C++ compiler.

CHAPTER 2 SETTING ENVIRONMENT VARIABLES IN SYSTEM BEFORE STARTING

This chapter describes environment variables in the system used to run the C/C++ compiler. (For information on setting variables, refer to the manual for the respective operating system.)

CHAPTER 3 C/C++ COMPILER OPERATION

This chapter describes the command function specifications.

CHAPTER 4 fcc911s COMMAND OBJECT PROGRAM STRUCTURE

This chapter explains about the information necessary for program execution.

CHAPTER 5 EXTENDED LANGUAGE SPECIFICATIONS

This chapter explains about the extended language specifications supported by the compiler. The limitations on compiler translation are also described in this chapter.

CHAPTER 6 EXECUTION ENVIRONMENT

This chapter describes the user program execution procedure to be performed in an environment where no operating system exists.

CHAPTER 7 LIBRARY OVERVIEW

This chapter outlines the C libraries by describing the organization of files furnished by the libraries and the relationship to the system into which the libraries are incorporated.

CHAPTER 8 LIBRARY INCORPORATION

This chapter describes the processes and functions for preparing for useing library.

CHAPTER 9 COMPILER-DEPENDENT SPECIFICATIONS

This chapter describes the specifications that vary with the compiler. Descriptions are related to JIS standard that are created based on ANSI standard.

CHAPTER 10 SIMULATOR DEBUGGER LOW-LEVEL FUNCTION LIBRARY

This chapter describes how to use the simulator debugger low-level function library.

APPENDIX

The appendix gives a list of types, macros, functions and variables provided by the libraries and describes the operations specific to the libraries (The APPENDIX A and APPENDIX B).

The list of the error message is described (The APPENDIX C).

The list of the reserved pragma directive is described (The APPENDIX D).

■ Grammar Books

For C or C++ language syntax and standard library functions, refer to commercially available standard compliant reference books.

■ Reference Books

THE C PROGRAMMING LANGUAGE
(Brian W.Kernighan & Dennis M.Ritchie)

Japanese edition entitled Programming Language C UNIX Type Programming Method and Procedure
(Translated by Haruhisa Ishida; Kyoritsu Shuppan)

American National Standard for Information Systems - Programming Language C, X3.159-1989

UNIX system User's Manual system V
(Western Electric Company, Incorporated)

UNIX system V Programmer Reference Manual
(AT&T Bell Laboratories)

User Reference Manual UTS/5 Release 0.1
(Western Electric Company, Incorporated and Amdahl Corporation)

UTS Command Reference Manual UTS/5 Release 0.1
(Western Electric Company, Incorporated and Amdahl Corporation)

The Annotated Reference Manual
(Addison-Wesley Publishing Company, Inc.)

The Programing Language C++ Third Edition
(Addison-Wesley Publishing Company, Inc.)

ISO/IEC 14882:1998 Programming languages -- C++

ISO/IEC 9899:1999 Programming languages -- C

- The contents of this document are subject to change without notice.
  Customers are advised to consult with sales representatives before ordering.
- The information, such as descriptions of function and application circuit examples, in this document are presented solely for the purpose of reference to show examples of operations and uses of FUJITSU SEMICONDUCTOR device; FUJITSU SEMICONDUCTOR does not warrant proper operation of the device with respect to use based on such information. When you develop equipment incorporating the device based on such information, you must assume any responsibility arising out of such use of the information. FUJITSU SEMICONDUCTOR assumes no liability for any damages whatsoever arising out of the use of the information.
- Any information in this document, including descriptions of function and schematic diagrams, shall not be construed as license of the use or exercise of any intellectual property right, such as patent right or copyright, or any other right of FUJITSU SEMICONDUCTOR or any third party or does FUJITSU SEMICONDUCTOR warrant non-infringement of any third-party's intellectual property right or other right by using such information. FUJITSU SEMICONDUCTOR assumes no liability for any infringement of the intellectual property rights or other rights of third parties which would result from the use of information contained herein.
- The products described in this document are designed, developed and manufactured as contemplated for general use, including without limitation, ordinary industrial use, general office use, personal use, and household use, but are not designed, developed and manufactured as contemplated (1) for use accompanying fatal risks or dangers that, unless extremely high safety is secured, could have a serious effect to the public, and could lead directly to death, personal injury, severe physical damage or other loss (i.e., nuclear reaction control in nuclear facility, aircraft flight control, air traffic control, mass transport control, medical life support system, missile launch control in weapon system), or (2) for use requiring extremely high reliability (i.e., submersible repeater and artificial satellite).
  Please note that FUJITSU SEMICONDUCTOR will not be liable against you and/or any third party for any claims or damages arising in connection with above-mentioned uses of the products.
- Any semiconductor devices have an inherent chance of failure. You must protect against injury, damage or loss from such failures by incorporating safety design measures into your facility and equipment such as redundancy, fire protection, and prevention of over-current levels and other abnormal operating conditions.
- Exportation/release of any products described in this document may require necessary procedures in accordance with the regulations of the Foreign Exchange and Foreign Trade Control Law of Japan and/or US export control laws.
- The company names and brand names herein are the trademarks or registered trademarks of their respective owners.

CONTENTS

CHAPTER 1

SOFTUNE C/C++ COMPILER

This chapter outlines the C/C++ compiler.
The C/C++ compiler is a language processor program which translates source programs written in C or C++ language into the assembly language for Fujitsu-provided various microcontroller units.

1.1 C/C++ Compiler Functions

When a source file represented in C or C++ language is described, the C/C++ compiler generates an assembler source file which is expressed in assembly language.

■ C/C++ Compiler Functions

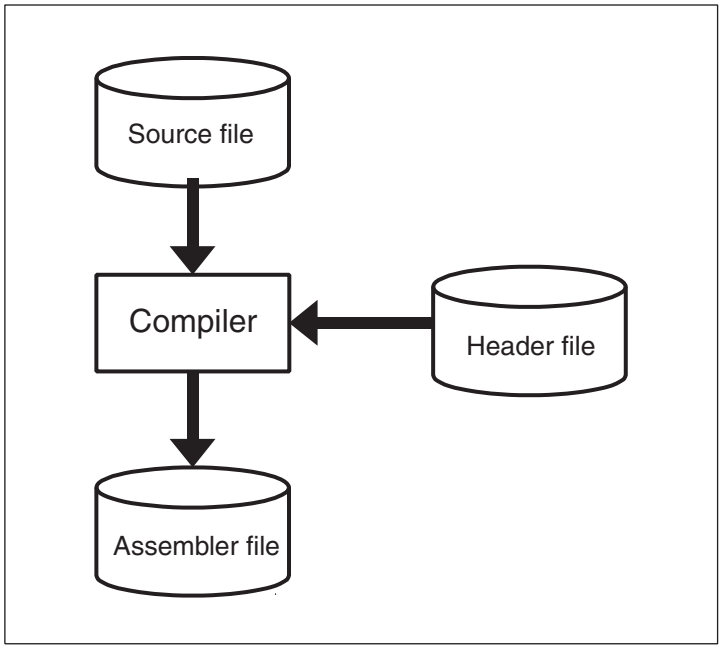
The C/C++ compiler generates an assembler source file using the procedures described below.

● [Compilation]

Compiling is performed by the compiler (cpcoms). The compiler translates C or C++ source files into assembler source files.

To operate the C/C++ compiler, use the fcc911s command. This command automatically sets the environment of the C/C++ compiler (such as the setting of an include directory) and controls compiling of C or C++ source files. Figure 1.1-1 shows the configuration of the C/C++ compiler.

Figure 1.1-1 Configuration of the C/C++ compiler



In the subsequent sections, the C/C++ compiler translation process is explained using commands. For the details of the command function specifications, see "CHAPTER 3 C/C++ COMPILER OPERATION".

1.2 Basic Process of Commands

This section describes the basic function of commands used in the C/C++ compiler. The C/C++ compiler uses the following command:

fcc911s ----- FR family command

■ Command Basic Process

The basic function of the command is to generate an absolute file from a C/C++ source file. The command recognizes files with the .c extension as C source files, and files with the .cc, .cpp, or .cxx extension as C++ source files.

A command use example is given below. > is the command prompt.

[Example]

```
> fcc911s -cpu MB91F154 file.c
```

When the above entry is made, the command assumes that file.c is a C source file. As far as no error is detected, an absolute file (file.abs) is generated in the current directory.

[Example]

```
> fcc911s -o outfile -cpu MB91F154 file.c
```

With the parameters set as indicated in the above example, the command generates an absolute file "outfile". By specifying options such as -o, the command can control the file generation process.

■ Options for Compiling Process Control

● [-P option]

When the -P option is specified, the command calls up the compiler only and performs preprocessing to generate a preprocessed C/C++ source file in the current directory. Files to be generated include files with extensions changed to .i for C and with extensions changed to .ipp for C++.

● [-S option]

When the -S option is specified, the command calls up the compiler to compile and thus generate an assembler source file in the current directory. The extension of the generated file is changed to .asm.

● [-c option]

When the -c option is specified, the command calls up the, compiler, and assembler and performs compilation, and assembling to generate an object file in the current directory. The extension of the generated file is changed to .obj.

● [-o option]

When the -o option is specified, the command generates the file specified in the command line as a result of processing.

1.3 C/C++ Compiler Basic Functions

The C/C++ compiler three functions are described below.
1) Header file search
2) Coordination with symbolic debugger
3) Optimization
The symbolic debugger is a support tool for analyzing a program created in C language or C++ language.

■ Header File Search

The header file can be acquired using the C or C++ program #include instruction. When the absolute pathname is specified, the header file enclosed within angular brackets (<>) is searched for in the directory defined by that pathname. When the absolute pathname is not specified, the compiler standard directory is searched.

The standard header file is supplied by the C/C++ compiler.

The header file enclosed by double quotation marks (") is searched for in a directory specified by the absolute pathname. If the absolute pathname is not specified, such a header file is searched for in a directory having a file containing a #include line. If the header file is not found in a directory having a file containing a #include line, the standard directory is searched next.

The -I option makes it possible to add a directory for header file search.

[Example]

```
> fcc911s -cpu MB91F154 -I ..\include file.c
```

When the above entry is made, the command searches for the header file enclosed within angular brackets in the order indicated below.

1. ..\include
2. Compiler standard include file directory

The header file enclosed by double quotation marks is searched for in the order indicated below.

1. Current directory having a file containing a #include line
2. ..\include
3. Compiler standard include file directory

The -I option can be specified a desired number of times. When it is specified two or more times, search operations are conducted in the specified order.

■ Coordination with Symbolic Debugger

When the -g option is specified, the compiler generates the debug information to be used by the symbolic debugger. When such information is generated, C/C++ language level debugging can be accomplished within the symbolic debugger. Two types of symbolic debuggers are available; simulator debugger and emulator debugger.

When the optimization option (-O[1-4]) is specified, debugging should be performed, noting the following points. When the optimization option is specified, the compiler attempts to ensure good code generation by changing the computation target position and eliminating computations that are judged to be unnecessary. To minimize the amount of data exchange with memory, the compiler tries to retain data within a register. It is therefore conceivable that a break point positioned in a certain line may fail to cause a break or that currently monitored certain address data may fail to vary with the expected timing. It also well to remember that the debug data will not be generated for an unused local variable or a local variable whose area need not be positioned in a stack as a result of optimization.

Take the above point into considerations when debugging is conducted.

■ Optimization

When the -O option is specified, the compiler generates an object subjected to general-purpose optimization.

CHAPTER 2

SETTING ENVIRONMENT VARIABLES IN SYSTEM BEFORE STARTING

This chapter describes environment variables in the system used to run the C/C++ compiler. (For information on setting variables, refer to the manual for the respective operating system.)
All environment variables can be omitted. For information on the supply style, refer to the C/C++ Compiler Installation Manual.
See Section "3.3 File Names and Directory Names", for details about the characters that can be used for the directories to be set up as environment variables in the Windows version.
[Setup Example]

```
set FETOOL=c:\Fujitsu MCU tool
```

For environment variable setup, do not use double quotation marks (").

2.1 FETOOL

Specify the directory where the development environment is installed.

■ FETOOL

[General Format 1 For UNIX OS]

setenv FETOOL Installation directory

[General Format 2 For Windows]

set FETOOL = Installation directory

The driver accesses the compiler, message file, include file, and other items via the path specified by FETOOL.

When FETOOL setup is not completed, the parent directory for the directory where the activated driver exists (the /.. position of the directory where the driver exists) is regarded as the installation directory.

No more than one directory can be specified.

[Example For UNIX OS]

```
setenv FETOOL /usr/local/softune6
```

[Example For Windows]

```
set FETOOL=c:\softune6
```

2.2 LIB911

Specify the directory for the library as LIB911.

■ LIB911

[General Format 1 For UNIX OS]

setenv LIB911 library directory [:directory 2...]

[General Format 2 For Windows]

set LIB911 = library directory [; directory 2...]

Specify the directory for the library to which linking is effected by default.

When LIB911 is not set up, the directory placed at the respective location relative to the FETOOL directory (%FETOOL%\lib\911) is regarded as the default library directory.

Two or more directories can be specified, separated by a delimiter. The delimiter is a colon ':' for the UNIX OS, and a semicolon ';' for Windows.

[Example For UNIX OS]

```
setenv LIB911 /usr/local/softune6/lib/911
```

[Example For Windows]

```
set LIB911=c:\softune6\lib\911
```

2.3 OPT911

Specify the directory for the default option file to be used by the fcc911s command as OPT911.

■ OPT911

[General Format 1 For UNIX OS]

setenv OPT911 Default option file directory

[General Format 2 For Windows]

set OPT911 = Default option file directory

Specify the directory for the default option file to be used by the driver.

When OPT911 is not set up, the directory at the corresponding relative position with respect to the FETOOL directory (%FETOOL%\lib\911) is regarded as the default option file directory.

No more than one directory can be specified.

[Example For UNIX OS]

```
setenv OPT911 /usr/local/softune6/lib/911
```

[Example For Windows]

```
set OPT911=c:\softune6\lib\911
```

2.4 INC911

Specify the directory for standard header files as INC911.

■ INC911

[General Format 1 For UNIX OS]

setenv INC911 Standard include directory

[General Format 2 For Windows]

set INC911 = Standard include directory

Specify the directory for standard header files. The directory specified as INC911 is regarded as the standard include-directory. When INC911 is not defined, the directory at the corresponding relative position with respect to the FETOOL directory (%FETOOL%\lib\911\include) is regarded as the standard header file directory.

Two or more directories can be specified, separated by a delimiter. The delimiter is a colon ':' for the UNIX OS, and a semicolon ';' for Windows.

[Example For UNIX OS]

```
setenv INC911 /usr/local/softune6/lib/911/include
```

[Example For Windows]

```
set INC911=c:\softune6\lib\911\include
```

2.5 TMP

Specify the directory for the temporary file to be used by the C compiler.

■ TMP

[General Format 1 For UNIX OS]

setenv TMP Temporary directory

[General Format 2 For Windows]

set TMP = Temporary directory

Specify the working directory for creating the temporary file to be used by the C compiler.

If TMP setup is not completed, the temporary file is created in the current directory.

No more than one directory can be specified.

[Example For UNIX OS]

```
setenv TMP /usr/tmp
```

[Example For Windows]

```
set TMP=c:\tmp
```

2.6 FELANG

Specify the code for messages.

■ FELANG

[General Format 1 For UNIX OS]

setenv FELANG Message code

[General Format 2 For Windows]

set FELANG = Message code

Specify the message code. The following codes can be specified.

ASCII:

Outputs messages in ASCII code. The generated messages are in English. Select this code for a system without a Japanese language environment.

EUC:

Outputs messages in EUC code. The generated messages are in Japanese.

SJIS:

Outputs messages in SHIFT JIS code. The generated messages are in Japanese. If FELANG setup is not completed, the ASCII code is considered to be selected.

[Example For UNIX OS]

```
setenv FELANG EUC
```

[Example For Windows]

```
set FELANG=SJIS
```

CHAPTER 3

C/C++ COMPILER OPERATION

This chapter describes the command function specifications.

3.1 Command Line

The command line format is shown below.
fcc911s [options] operands

■ Command Line

Options and operands can be specified in the command line. Options and operands can be specified at any position within the command line. Two or more options and operands can be specified. Options can be omitted.

Option and operand entries are to be delimited by a blank character string. The command recognizes the options and operands in the order indicated below.

1. An entry beginning with a hyphen (-) is first recognized as an option. The subsequent character string is interpreted to determine the option type.
2. As regards an option having an argument, the subsequent character string is regarded as the argument.
3. The remaining entries in the command line are recognized as operands.

[Example]

```
>fcc911s file1.c -S -I \home\myincs file2.c -cpu MB91F154
```

At first, -S and -I are regarded as options. Since the -I option has an argument, the subsequent character string \home\myincs is regarded as the argument. The remaining entries (file1.c and file2.c) are regarded as operands.

Options:

```
-S, -I \home\myincs
```

Operands:

```
file1.c, file2.c
```

■ Command Process

The command calls a compiler, assembler, and linker for each input file in the order they are specified to compile, assemble, and link. The result of each module is output to a file having the same file name as an input file name with its extension replaced by .obj.

The linking result, unless changed by the -o option, is output to files with .abs extensions to which the extensions of the files first specified are changed.

[Example]

```
>fcc911s file1.c file2.c file3.c -cpu MB91F154
```

Files named file1.c, file2.c, and file3.c are subjected to compiling, assembling, and linking so that files named file1.abs is generated.

3.2 Command Operands

One or more input files can be specified as operands.

■ Command Operands

The command determines the file type according to the input file extension and performs an appropriate process to suit the file type. The extension cannot be omitted.

File specifying:

C source files, preprocessed C source files, C++ source files, preprocessed C++ source files, assembler source files, and object files.

File extension:

The relationship between input file extensions and command processes is shown in Table 3.2-1.

Table 3.2-1 Relationship between Extensions and Command Processes

Extension	Command Process
.c	Files having this extension are compiled and subjected to subsequent processes as C source files.
.i	Files having this extension are compiled and subjected to subsequent processes as C source files that have been preprocessed.
.cc	Regard files as the C++ source files to perform processing subsequent to compiling.
.cpp	Regard files as the C++ source files to perform processing subsequent to compiling.
.cxx	Regard files as the C++ source files to perform processing subsequent to compiling.
.ipp	Regard files as the preprocessed C++ source files to perform processing subsequent to compiling.
.asm	Files having this extension are assembled and subjected to subsequent processes as assembler source files that have been compiled.
.obj	Files having this extension are linked and subjected to subsequent processes as object files that have been assembled.
.abs	The file having this extension is regarded as a linked absolute file, and an error output is generated. No absolute file can be specified.

Note, however, that the associated process may be inhibited depending on the option specifying.

[Example]

```
>fcc911s file1.c file2.asm -cpu MB91F154
```

The file1.c file is compiled and assembled, and the file2.asm file is assembled. Linking is then performed to generate the file1.abs file.

3.3 File Names and Directory Names

The following characters are applicable to file names and directory names.

■ File Names and Directory Names

● Windows

Alphanumeric characters, symbols except \, /, :, *, ?, ", <, >, and |, Shift-JIS kanji codes, and Shift- JIS 1-byte kana codes. Enclose a long file name in double quotation marks (") when the file is specified as an option or operand. The double quotation marks, however, cannot be used when the file is specified as an environment variable.

● UNIX OS

Underbar "_" and alphanumeric characters (However, the first character must be the underbar or alphabetical character).

● Module name

A module name is generated from a file name. The characters that can be used for a module name are an underbar "_" and alphanumeric characters. (The first character of the name must be an underbar "_" and an alphabetic character.) When other characters are used for a file name, a character that cannot be used for a module name is converted to an underbar "_". A file name that will be identical to a module name after conversion should not be used.

[Example]

```
#abc.c and -abc.c will have the same module name (_abc).
```

3.4 Command Options

The command options are explained below.

■ Option Syntax

The option consists of a hyphen (-) and one or more characters following the hyphen. Some options have an argument. A blank character string must be positioned between an option and an argument. The command options cannot be grouped for purposes of specifying. Grouping is a technique of specifying which, for instance, uses a -Sg form to specify both the -S option and -g option.

■ Multiple Specifying of Same Option

If the same option is specified more than one time, only the last-specified option in the command line is assumed to be valid.

[Example]

```
>fcc911s -o outfile file.c -o outobj -cpu MB91F154
```

The name of a file to be output is outobj. The options listed below can be specified more than once for the same command and are different in meaning each time.

- Options that are significant when specified more than one time

-D, -f, -I, -INF, -K, -L, -l, -ra, -ro, -sc, -T, -U, -x, -Y

When the above options are specified more than one time, see details of options.

■ Position within Command Line

The option's position within the command line does not have a special meaning. Options are interpreted in the same manner no matter where in the command line they are specified.

[Example]

```
1) >fcc911s -C -E file1.c file2.c -cpu MB91F154
2) >fcc911s file1.c -E file2.c -C -cpu MB91F154
```

The same processing operations are performed for cases 1 and 2.

■ Exclusiveness and Dependency

Options may be mutually exclusive or mutually dependent. For information on the exclusivity and dependence of options, see option descriptions.

■ Case Sensitiveness

As regards the options, their upper-case and lower-case characters are different from each other. For example, the -O option is different from the -o option. However, the upper- and lower-case characters of suboptions are not differentiated from each other. For example, the -K eopt option is considered in the same as the -K EOPT option. The suboptions are the character strings that follow the -K option or -INF option.

3.4.1 List of Command Options

When executed without argument specifying, the command outputs an option list to the standard output.
Table 3.4-1 and Table 3.4-2 list the command options. Options listed in the tables are recognized by the command.

■ List of Command Options

Table 3.4-1 List of Command Options (1 / 3)

Specifying Format	Function
-align {FUNC4 | FUNC8 }	Specifies that branch labels are to be aligned.
-B	Allows a C++-style comment //.
-C	Leaves a comment in the preprocessing result.
-c	Performs processing up to assembling and output the result to .obj.
-cif filename	Specifies the CPU information file.
-cmsg	Outputs the message of the end of compiling process to the standard output.
-CO	Generates the compatible object.
-cpu MB number	Specifies the MB number of the CPU used.
-cwno	Sets the end code to 1 when a warning occurs.
-D name [= [tokens]]	Defines the macro name.
-E	Performs preprocessing only and outputs the result to the standard output.
-e name	Specifies the entry of a program.
-f filename	Specifies the option file.
-g	Inserts the debugging information into the object.
-H	Outputs the acquired header file pathname to the standard output.
-help	Outputs the option list to the standard output.
-HH	Outputs the acquired header file pathname to the standard output and generates the object.
-I dir	Specifies the directory of the header file.
-INF LIST	Generates the assemble list.
-INF LONGMESSAGE	Details a message at translating.
-INF {SRCIN | LINENO}	Insert the C source into the assembler source.
-INF STACK [=filename]	Generates the information file of the used stack.

Table 3.4-1 List of Command Options (2 / 3)

Specifying Format	Function
-J {a | c | e}	Specifies the language specification level.
-K {A1 | A4}	Specifies the minimum boundary alignment value for static data.
-K BP	Specifies to generate BP relative access instructions.
-K CNC	Specifies the method of handling an external symbol in the CONST section.
-K {DCONST | FCONST}	Specifies the type of real constants without a suffix.
-K EOPT	Specifies the optimization of the evaluation order of arithmetic operations.
-K LIB	Recognizes the standard function operation and implements in-line expansion/substitution for other functions.
-K LONGLONG	Treats the long long type as the 8-byte integer type.
-K MERGESTRING	Merges substances of the same character string literal.
-K NOALIAS	Specifies the optimization on the presumption that differing pointers do not reference the same memory area.
-K NOCALL21	Specifies not to use the CALL21 instruction.
-K NOFPU	Specifies not to use the FPU.
-K NOINTLIB	Disables inline expansion of interrupt related functions.
-K NOSAVEFREG	Specifies not to restore FPU registers in an interrupt function.
-K NOUNROLL	Inhibits loop unrolling.
-K NOVOLATILE	Does not consider __io qualified variables to be volatile.
-K {SCHEDULE | NOSCHEDULE}	Specifies the running scheduler.
-K {SHORTADDRESS [= {CODE | DATA}] | LONGADDRESS [= {CODE | DATA}] }	Specifies the method for handling external symbols.
-K {SIZE | SPEED}	Selects optimization based on the size and execution speed.
-K {UBIT | SBIT}	Specifies the plain int bit field as the signed bit field.
-K {UCHAR | SCHAR}	Specifies the plain char type as the signed char type.
-kanji {SJIS | EUC}	Specifies the kanji code used in a program.
-l lib1 [, lib2...]	Specifies the library file name.
-L path1 [,path2...]	Specifies the library path.
-m	Outputs a map file at the time of linking.
-O level	Specifies the optimization level.
-o pathname	Outputs the result to the pathname.

Table 3.4-1 List of Command Options (3 / 3)

Specifying Format	Function
-P	Performs preprocessing only and outputs the result to .i or .ipp.
-ra name = start/end	Specifies the RAM area.
-ro name = start/end	Specifies the ROM area.
-S	Performs processes up to compiling and outputs the result to .asm.
-s defname = newname [, attr [, address]]	Changes the section name.
-sc param	Specifies the section arrangement.
-startup file	Specifies the startup file name.
-T item, arg1 [arg2...]	Passes arguments to the tool.
-U name	Cancels the macro name definition.
-V	Outputs the executed compiler tool version information to the standard output.
-varorder {SORT|NORMAL}	Specifies the allocation type of static variables.
-w level	Specifies the output level of warning messages.
-x func [, func2...]	Specifies the inline expansion of functions.
-xauto [size]	Specifies the inline expansion of the functions whose logical line count is less than the value specified for the size line.
-Xdof	Inhibits the default option file read operation.
-Y item, dir	Changes the item position to dir.

Table 3.4-2 List of Command Options (for C++ Source)

Specifying Format	Function
-t {none | used | all | local}	Specifies the type of template instantiation.
--alternative_tokens	Enables an alternative keyword.
--no_auto_instantiation	Suppresses automatic instantiation of a template.
--old_for_init	Uses the ANSI or earlier specifications for the scope of declaration in the for initialize expression.
--for_init_diff_warning	Controls a warning on different interpretations between --old_for_init and --new_for_init.
--suppress_vtbl	Suppresses definition of a virtual function table.
--force_vtbl	Forces definition of a virtual function table.

3.4.2 List of Command Cancel Options

The listed options are used to cancel command options on an individual basis. The cancel options for the command are listed in Table 3.4-3 and Table 3.4-4.

■ List of Command Cancel Options

Table 3.4-3 List of Command Cancel Options (1 / 2)

Specifying Format	Function
-INF NOLINENO	Cancels the LINENO suboption.
-INF NOLIST	Cancels the LIST suboption.
-INF NOSRCIN	Cancels the SRCIN suboption.
-INF NOSTACK	Cancels the STACK suboption.
-INF SHORTMESSAGE	Cancels the LONGMESSAGE suboption.
-K ALIAS	Cancels the NOALIAS suboption.
-K CALL21	Cancels the NOCALL21 suboption.
-K FPU	Cancels the NOFPU suboption.
-K INTLIB	Cancels the NOINTLIB suboption.
-K NOBP	Cancels the BP suboption.
-K NOCNC	Cancels the CNC suboption.
-K NOEOPT	Cancels the EOPT suboption.
-K NOLIB	Cancels the LIB suboption.
-K NOLONGLONG	Cancels the LONGLONG suboption.
-K NOMERGESTRING	Cancels the MERGESTRING suboption.
-K SAVEFREG	Cancels the NOSAVEFREG suboption.
-K UNROLL	Cancels the NOUNROLL suboption.
-K VOLATILE	Cancels the NOVOLATILE suboption.
-Xalign	Cancels the -align option.
-XB	Cancels the -B option.
-XC	Cancels the -C option.
-Xcmsg	Cancels the -cmsg option.
-XCO	Cancels the -CO option.
-Xcwno	Cancels the -cwno option.

Table 3.4-3 List of Command Cancel Options (2 / 2)

Specifying Format	Function
-Xe	Cancels the -e option.
-Xf	Cancels the -f option.
-Xg	Cancels the -g option.
-XH	Cancels the -H option.
-Xhelp	Cancels the -help option.
-XHH	Cancels the -HH option.
-XI	Cancels the -I option.
-XL	Cancels the -L option.
-Xm	Cancels the -m option.
-Xo	Cancels the -o option.
-Xra	Cancels the -ra option.
-Xro	Cancels the -ro option.
-Xs	Cancels the -s option.
-Xsc	Cancels the -sc option.
-Xstartup	Cancels the -startup option.
-XT item	Cancels the -T item specifying.
-XV	Cancels the -V option.
-Xx	Cancels the -x option.
-Xxauto	Cancels the -xauto option.
-XY item	Cancels the -Y item specifying.

Table 3.4-4 List of Command Cancel Options (For C++ Source)

Specifying Format	Function
--no_alternative_tokens	Cancels the --alternative_tokens option.
--auto_instantiation	Cancels the --no_auto_instantiation option.
--new_for_int	Cancels the --old_for_init option.
--no_for_int_diff_warning	Cancels the --for_init_diff_warning option.

3.5 Details of Options

This section details the options.

■ Translation control related options

The translation control related options are related to preprocessing, compiler, assembler, and linker call control.

■ Preprocessor related options

The preprocessor related options are related to preprocessing operations

■ Data output related options

The data output related options are related to the command, preprocessing, and compiler data outputs.

■ Language specification related options

The language specification related options are related to the specification of the language to be recognized by the compiler.

■ Optimization related options

The optimization related options are related to the optimization to be effected by the compiler.

■ Output object related options

The output object related options are related to the output object format.

■ Debug information related options

The debug information related options are related to the debug information to be referenced by the symbolic debugger.

■ Command related options

The command related options are related to the other tools recalled by commands.

■ Linkage related options

The linkage related options are related to linkage.

■ Option file related options

The option file related options are associated with option files.

3.5.1 Translation Control Related Options

The translation control related options are related to preprocessor, compiler, assembler, and linker call control.

■ Translation Control Related Options

The priorities of the translation control related options are defined as follows. They are not related to the order of specifying.

-E > -P > -S > -c

The translation control related option exclusiveness is shown in Table 3.5-1.

Table 3.5-1 Translation Control Related Option Exclusiveness

Specified Option	Option Invalidated
-E	-S and -c
-P	-S and -c
-S	-c
-c	None

If the -E and -P options are specified simultaneously, see the explanation below.

● -E Option

-E Option subjects all files to preprocessing and outputs the result to the standard output. The output result contains the preprocessing instruction generated by the preprocessor, which is necessary for the compiler. The information targets for the preprocessing instruction generated by the preprocessor are the #line and #pragma instructions. If the -P option is specified together with the -E option, the preprocessing instruction generated by the preprocessor is inhibited.

If the input file is not a C source file or C++ source file, the -E option does not process anything.

[Example]

```
>fcc911s -E -cpu MB91F154 sample.c
```

The sample.c preprocessing result is output to the standard output.

● -P Option

-P option subjects a C source file or C++ source file to preprocessing and outputs the result to the file whose extension is changed to .i or .ipp. Unlike the cases where the -E option is specified, the output result does not contain the preprocessing instruction generated by the preprocessing. If the input file is not a C source file, the -P option does not process anything.

[Example]

```
>fcc911s -P -cpu MB91F154 sample.c
```

The sample.c preprocessing result is output to the sample.i.

[Example]

```
>fcc911s -P -cpu MB91F154 sample.cpp
```

The result of preprocessing for sample.cpp is output to sample.ipp.

● -S Option

-S option performs processes up to compiling and outputs the resultant assembler source to file extension changed to .asm.

If the input file is neither a C source file nor a preprocessed C source file, the -S option does not process anything.

[Example]

```
>fcc911s -S -cpu MB91F154 sample.c
```

The sample.c preprocessing and compiling process result are output to the sample.asm.

● -c Option

Performs processes up to assembling and outputs the resultant object to file extension changed to .obj. If the input file is an object file, the -c option does not process anything.

[Example]

```
>fcc911s -c -cpu MB91F154 sample.c
```

The results of preprocessing, compiling, and assembling to sample.c are output to sample.obj.

The relationship among file types and processes for translation control related options is shown in Table 3.5-2.

Table 3.5-2 Relationship Among File Types and Processes for Translation Control Related Options

Option File Type	-E	-P	-S	-c	Nothing Specified
C/C++ source file	P	P	P and C	P, C and A	P, C, A and L
Preprocessed C/C++ source file	-	-	C	C and A	C, A and L
Assembler source file	-	-	-	A	A and L
Object file	-	-	-	-	L

P: Preprocessing
C: Compiling
A: Assembling
L: Linking

[Example]

```
>fcc911s -E file1.c file2.i -cpu MB91F154
```

Subjects files named file1.c to preprocessing and outputs the result to the standard output. No processing is performed for file2.i.

```
>fcc911s -S file1.c file2.i file3.asm -cpu MB91F154
```

Subjects a file named file1.c to preprocessing and compiling and a file named file2.i to compiling. Performs nothing for a file named file3.asm. As a result, files named file1.asm and file2.asm are generated in the current directory.

3.5.2 Preprocessing Related Options

This section deals with the options related to preprocessing operations. If the preprocessing is not called, the preprocessing related options are invalid.

■ Preprocessing Related Options

The preprocessing related options are detailed below.

● -B option and -XB option

The -B option regards items following // in the C source as comments.

The -XB option cancels the -B option. This option is ignored even if it is specified in the C++ source.

When the -Jc option is specified, -B option is disabled.

[Example]

```
Input:     void func()
           {
           //empty function
           }
Operation: fcc911s -S -B -cpu MB91F154 sample.c
```

● -C option and -XC option

-C options are all comments except those which are in the preprocessing instruction line and will be retained as the preprocessing result. If the option is not specified, the comments are replaced by one blank character.

The -XC option cancels the -C option.

[Example]

```
Input      /* Comment */
           void func(void) { }
Operation: fcc911s -C -E -cpu MB91F154 sample.c
Output     # 1 "test5.c"
           /* Comment */
           void func(void) { }
```

● -D name [=[tokens]] Option

Defines the macro name with the tokens used as the macro definition. This option is equal to the following #define instruction.

#define name tokens

If =tokens entry is omitted, the value "1" is given as the tokens value. If the tokens entry is omitted, the specified lexis is deleted from the source file. The error related to the -D option is the same as the error related to the #define instruction.

This option can be specified more than one time.

[Example]

```
>fcc911s -D os=m -D sys file.c -cpu MB91F154
```

In a file named file.c, processing is conducted on the assumption that the macro definitions for os and sys are m and 1, respectively.

● -H option and -XH option

-H option outputs the header file pathnames acquired during preprocessing to the standard output. The pathnames are sequentially output, one for each line, in the order of acquisition. If there are any two exactly the same pathnames, only the first one will be output. When this option is specified, the command internally sets up the -E option to subjects all files to preprocessing only. However, the preprocessing result will not be output. The -XH option cancels the -H option.

[Example]

```
Input      #include <stdio.h>
           #include "head.h"
Operation: fcc911s -H -cpu MB91F154 sample.c
Output     /usr/softune5/lib/911/include/stdio.h
           ./head.h
```

● -HH option and -XHH option

-HH option outputs the header file pathnames acquired during preprocessing to the standard output. The pathnames are sequentially output, one for each line, in the order of acquisition. If there are any two exactly the same pathnames, only the first one will be output. The -HH option differs from -H option, and generates the object file according to the specification of translation control option (-E, -P, -S, -c). The -XHH option cancels the -HH option.

[Example]

```
Input:     #include <stdio.h>
           #include "head.h"
Operation: fcc911s -HH -S -cpu MB91F154 sample.c
Output:    /usr/softune6/lib/911/include/stdio.h
           ./head.h
```

● -I dir option and -XI option

Changes the manner of header file search so that the directory specified by dir will be searched prior to the standard directory.

The standard directory is ${INC911} for the fcc911s command.

This option can be specified more than one time. The search will be conducted in the order of specifying. When this option is specified, the header file search will be conducted in the following directories in the order indicated below.

[Header file enclosed within angular brackets (< >)]

1. Directory specified by the -I option
2. Standard directory

[Header file enclosed by double quotation marks (")]

1. Directory having a file containing the #include line
2. Directory specified by the -I option
3. Standard directory

If a header file is specified by specifying its absolute path name, only the directory specified by the specified absolute path name will be searched. If any nonexistent directory is specified, this option is invalid.

The -XI option cancels the -I option.

● -U name option

Cancels the macro name definition formulated by -D. This option is equivalent to the following #undef instruction.

#undef name

If the same name is specified by the -D and -U options, the name definition will be canceled without regard to the order of option specifying. This option can be specified more than one time.

The error related to the -U option is the same as the error related to the #undef instruction.

[Example]

```
>fcc911s -U m -D n -D m file.c -cpu MB91F154
```

This will cancel the macro m definition formulated by the -D option.

3.5.3 Data Output Related Options

This section deals with the options related to the command, preprocessor, and compiler data outputs.

■ Data Output Related Options

- -cmsg Option

  Outputs the compiling process completion message.

  [Example]

```
Operation: fcc911s -cmsg -S -cpu MB91F154 sample.c
Output:    COMPLETED C Compile, FOUND NO ERROR : sample.c
```

- -cwno Option

  Set the end code to 1 when a warning-level error occurs. When this option is not given, the end code is 0.

- -help option and -Xhelp option

  Outputs the option list to the standard output.

  The -Xhelp option cancels the -help option.

  [Example]

```
>fcc911s -help
```

  Various command option lists are output to the standard output.

- -INF LINENO option and -INF NOLINENO option

  -INF LINENO option inserts C or C++ source file line numbers into the assembler source file as comments. The LINENO suboption cannot be specified simultaneously with the SRCIN suboption.

  The NOLINENO suboption cancels the LINENO suboption.

[Example]

```
Input:     void func(void) { )
Operation: fcc911s -INF lineno -S -cpu MB91F154 sample.c
Output     _func:
                   ST    RP, @-SP
                   ENTER #4
           ;;;;    sample.c, line 1
           L_func:
                   LEAVE
                   LD    @-SP+, RP
                   RET
```

● -INF LIST option and -INF NOLIST option

-INF LIST option outputs the assemble list. Generates a file in the current directory. The name of the generated file is determined by changing the source file name extension to .lst.

Since the assemble list is generated at assembling, it is not generated when assembling is not conducted.

For the details of the assemble list, refer to the Assembler Manual.

The NOLIST suboption cancels the LIST suboption.

[Example]

```
>fcc911s -INF list -c -cpu MB91F154 sample.c
```

The sample.c preprocessing, compiling, and assembling process result are output to the sample.obj, and the resulting assemble list is output to the sample.lst.

● -INF SRCIN option and -INF NOSRCIN option

-INF SRCIN option inserts a C or C++ source file into the assembler source file as a comment. The NOSRCIN suboption cancels the SRCIN suboption.

The SRCIN suboption cannot be specified simultaneously with the LINENO suboption.

[Example]

```
Input:     void func(void) { }
Operation: fcc911s -INF srcin -S -cpu MB91F154 sample.c
Output:    _func:
                   ST    RP, @-SP
                   ENTER #4
           ;;;;    void func(void) { }
           L_func:
                   LEAVE
                   LD    @-SP+, RP
                   RET
```

● -INF STACK [=file] option and -INF NOSTACK option

-INF STACK [=file] option outputs the stack use amount data. Generates the specified file in the current directory and outputs the stack use amount data. If no file is specified, the information in all the simultaneously compiled files is output into files whose names are determined by changing the source file extensions to .stk.

For stack use amount data utilization procedures and data file specifications, refer to the SOFTUNE C/C++ analyzer Manual.

The NOSTACK suboption cancels the STACK suboption.

[Example]

```
Input:     extern void sub(void);
           void func(void){sub();}

Operation: fcc911s -INF stack -S -cpu MB91F154 sample.c
Output:    @sample.c
           #  E=Extern   S=Static   I=Interrupt
           #   {Stack}    {E|S|I}    {function name} [A]
           #      ->       {E|S}      {call function}
           #               ...
           #
                   8        E         _func
                   ->       E         _sub
```

● -o pathname option and -Xo option

-o pathname option uses the pathname as the output file name. If this option is not specified, the default for the employed file format is complied with.

The -Xo option cancels the -o option.

[Example]

```
>fcc911s -o output.asm -S -cpu MB91F154 sample.c
```

The sample.c preprocessing and compiling process result are output to the output.asm.

● -V option and -XV option

-V option outputs the version information about each executed compiler tool to the standard output. The -XV option cancels the -V option.

[Example]

```
FR Family SOFTUNE C/C++ Compiler V60L01
ALL RIGHTS RESERVED, COPYRIGHT (C) FUJITSU LIMITED 1998
LICENSED MATERIAL - PROGRAM PROPERTY OF FUJITSU LIMITED
```

● -w [level] option

-w [level] option specifies the output level of warning-type diagnostic messages. The level can be specified between 0 and 8 but there is no difference between levels 1 and 8. This level specification is provided to ensure compatibility between the SOFTUNE C compiler and option.

When level 0 is specified, no warning messages will be generated. If the level description is omitted, level 0 is assumed. When the -w [level] option itself is omitted, -w 1 is applied.

For the details of diagnostic messages, see section "3.7 Messages Generated in Translation Process".

[Example]

```
Input:     const unsigned int a=-1;
Operation: fcc911s -INF srcin -S -cpu MB91F154 sample.c
Output:      *** sample.c(1) W1068B: warning: integer conversion resulted in a change of sign
Operation: fcc911s -w -S -cpu MB91F154 sample.c
Output:    (none)
```

● --for_init_diff_warning option and --no_for_init_diff_warning option

The --for_init_diff_warning option is valid only when the input file is a C++ source file. This option is ignored when the --old_for_init option is specified. The --for_init_diff_warning option specifies that a warning message is output if the initialize statement in the for statement is interpreted differently between the old and new C++ specifications.

The --no_for_init_diff_warning option cancels the --for_init_diff_warning option.

[Example]

```
Input:
     void func(int);
     int i;
     main()
     {
       for(int i=0; i<10; i++)
           func(i);
       func(i);
     }
Operation: fcc 911s -S -cpu MB91F154 --for_init_diff_warning sample.cpp
Output: *** sample. cpp(7) W1780B: warning: reference is to variable "i"(declared at line 2)
        -- under old for-init scoping rules it would have been variable "i"(declared at line 5)
```

● -INF LONGMESSAGE option and -INF SHORTMESSAGE option

The -INF LONGMESSAGE option details the diagnostic message of the compiler. The '^' mark indicates where an error occurs. Note that an incorrect location may be indicated depending the error type. It should also be noted that if the diagnostic message is displayed in a proportional font, the indicated location seems displaced.

The -INF SHORTMESSAGE option cancels the -INF LONGMESSAGE option.

[Example]

```
Input:
     int a b;
     void func( )
     {
     }
Operation: fcc911s -S -cpu MB91F154 -INF LONGMESSAGE sample.c
Output:    *** sample.c(1) E4065B: expected a ";"
             int a b;
                   ^
```

3.5.4 Language Specification Related Options

This section deals with the options related to the specifications of the language to be recognized by the compiler.

■ Language Specification Related Options

● -J {a | c | e} Option

Specifies the language specification level to be interpreted by the compiler (preprocessor included).

When -Ja is specified, interpretation is conducted in compliance with the ANSI standard including expansion specifications.

When -Jc is specified, interpretation is conducted in strict compliance with the ANSI standard. In response to the expansion specifications, a warning message is output.

Moreover, the following functions was disabled.

-B option (Allow a C++-style comment //)

When the -Je option is specified, the C++ source file is interpreted as being based on the EC++ language specifications. In addition to a normal warning, a warning message is output at description of specifications outside the range of the EC++ language specifications. The C source file has exactly the same meaning as that when the -Ja option is specified.

If this option is not specified, -Je applies.

● -K {DCONST|FCONST}Option

When the FCONST suboption is specified, a floating-point constant whose suffix is not specified will be handled as a float type. When the DCONST suboption is specified, a floating-point constant whose suffix is not specified will be handled as a double type. If neither of the above two suboptions is specified, -K DCONST applies.

[Example]

```
Input:     extern float f1,f2;
           void func(void) { f1=f2+1.0;}
Operation: fcc911s -K fconst -cpu MB91F154 -S sample.c
Output:    _func:
                   ST     RP, @-SP
                   ENTER  #4
                   LDI:32 #_f2, R12
                   LD     @R12, R4
                   LDI    #H'3F800000, R5
                   CALL32 _ _addf, R12
                   LDI:32 #_f1, R12
                   ST     R4,@R12
           L_func:
```

```
                LEAVE
                LD      @SP+, RP
                RET
```

● -K NOINTLIB option and -K INTLIB option

The NOINTLIB suboption calls a normal function without effecting in-line expansion of an interrupt related function (__DI(), __EI(), __set_il()). The INTLIB suboption cancels the NOINTLIB suboption.

[Example]

```
Input:     void func(void){ _ _DI( );}
Operation: fcc911s -K nointlib -cpu MB91F154 -S sample.c
Output:    _func:
                ST      RP, @-SP
                ENTER   #4
                CALL32  _ _ _DI, R12
           L_func:
                LEAVE
                LD      @SP+, RP
                RET
```

● -K NOVOLATILE option and -K VOLATILE option

The NOVOLATILE suboption does not recognize a __io qualifier attached variable as a volatile type. Therefore, __io qualifier attached variables will be optimized. The VOLATILE suboption cancels the NOVOLATILE suboption.

[Example]

```
>fcc911s -K novolatile -S -O -cpu MB91F154 sample.c
```

When an __io qualifier attached variable is processed in sample.c, it is not handled as a volatile qualifier attached variable, but is treated as the optimization target.

● -K {UCHAR|SCHAR}option

-K {UCHAR|SCHAR}option specifies whether or not to treat the char type most significant bit as a sign bit. When the UCHAR suboption is specified, the most significant bit will not be treated as a sign bit. When the SCHAR suboption is specified, the most significant bit will be treated as a sign bit.

If neither of the above two suboptions is specified, -K UCHAR applies.

[Example]

```
Input:     extern  int     data;
           char            c= -1;
           void func (void) {data=c;}
Operation: fcc911s -K schar -cpu MB91F154 -S sample.c
Output:            LDI:32   #_c,R12
                   LDUB     @R12,R0
                   EXTSB    R0                ;Code-extended
```

```
                    LDI:32   #_data,R12
                    ST       R0,@R12
```

● -K {UBIT|SBIT} option

Specifies whether or not to treat the most significant bit as a sign bit in situations where the char, short int, or long int type is selected as the bit field. When the UBIT suboption is specified, the most significant bit will not be treated as a sign bit. When the SBIT suboption is specified, the most significant bit will be treated as a sign bit.

If neither of the above two suboptions is specified, -K UBIT applies.

[Example]

```
Input      extern int  data;
           struct tag{ int bf:1;} st = {-1};
           void func(void) { data = st.bf;}
Operation: fcc911s -K sbit -cpu MB91F154 -S sample.c
Output:            LDI32  #_st, R12
                   LDUB   @R12, R0
                   EXTSB  R0                  ; Code-extended
                   ASR    #7, R0
                   LDI:32 #_data, R12
                   ST     R0, @R12
```

● -kanji {SJIS|EUC} option

If Japanese are entered in a program, the code system for the used Japanese is specified.

Japanese including 1-byte kana can be entered in program comments and character strings. The compiler identifies the code system for Japanese description based on this option. SJIS means that the Shift JIS code system is used, and EUC means that the EUC code system is used. When this option is omitted, -kanji EUC is used for Solaris, and -kanji SJIS is used for HP-UX and Windows.

● -K LONGLONG option and -K NOLONGLONG option

The LONGLONG suboption treats the following type as 8-byte integer type.

long long int

signed long long int

unsigned long long int

Moreover, the following specification is effective.

- Pre-defined macro __LONGLONG__ is defined as 1.
- In lib/911/include/limits.h, the macro of LONG_LONG_MIN, LONG_LONG_MAX, and ULONG_LONG_MAX is effective.
- In lib/911/include/builtin.h, intrinsic function __muls() and __mulu() is effective.

The following limitation is in 8-byte integer type.

- __io type qualifier cannot be specified for the variable of 8-byte integer type.
- 8-byte integer type cannot be specified for bit field member's type.

The NOLONGLONG suboption cancels the LONGLONG suboption.

- --alternative_tokens option and --no_alternative_tokens option

  The --alternative_tokens option enables alternative representation. For the specifications for alternative representation, refer to ISO/ICE 14882:1998.

  The --no_alternative_tokens option cancels the --alternative_tokens option.

- --old_for_init option and --new_for_init option

  The --old_for_init option makes a change so that the scope of variables declared in the initialize statement in the for statement is interpreted as being within the old specifications.

  The --new_for_init option cancels the --old_for_init option.

  [Example]

```
Input:
     extern void func(int);
     int main( )
     {
        for(int i=0; i<10; i++)
              func(i);
        func(i);               // Possible only for old specifications
        return 0;
     }
Operation: fcc911s -S --old_for_init sample.cpp -cpu MB91F154
```

3.5.5 Optimization Related Options

This section deals with the options related to optimization by the compiler.

■ Optimization Related Options

● -K SIZE option

Selects an appropriate optimization combination with emphasis placed upon the object size. The selected options in the fcc911s command are indicated below.

- -O 3
- -K EOPT
- -K NOUNROLL
- -K SHORTADDRESS

To change the above combination, specify, for example, the -O1 option following the -SIZE sub-option.

The -K SIZE option not only offers the optimization combination selection function, but also makes it possible to issue a generation instruction for object size minimization and effect object pattern switching.

● -K SPEED option

Selects an appropriate optimization combination with emphasis placed upon the generated object execution speed. The selected options in the fcc911s command are indicated below.

- -O 4
- -K SHORTADDRESS
- -align FUNC4

To change the above combination, specify, for example, the -K LONGADDRESS option following the SPEED sub-option.

The -K SPEED option not only offers the optimization combination selection function, but also makes it possible to issue a generation instruction for execution speed maximization and effect object pattern switching.

● -O [level]option

-O [level] specifies the optimization level. Levels 0, 1, 2, 3, and 4 can be specified. The higher the optimization level, the shorter the generated object execution time but the longer the compilation time. Note that each optimization level contains lower optimization level functions.

One of the following levels is to be specified. When no level is specified, -O2 applies.

- 0(Optimization level 0): No optimization will be effected. This level is equivalent to cases where the -O is not specified.
- 1(Optimization level 1): Optimization will be effected in accordance with detailed analyses of a program flow. In addition, the instruction scheduling will be conducted.
- 2(Optimization level 2): The following optimization feature is exercised in addition to the feature provided by optimization level 1.
- Loop unrolling: Loop unrolling is performed to increase the execution speed by decreasing the loop count when loop- count detection is possible. However, it tends to increase object size. Therefore, this optimization should not be used in situations where object size is important.

[Example]

```
(Before unrolling)
                  for(i=0;i<3;i++) { a[i]=0;}
(After unrolling)
                  a[0]=0;
                  a[1]=0;
                  a[2]=0;
```

- 3(Optimization level 3): The following optimization features are exercised in addition to the features provided by optimization level 2.
  - Loop unrolling (extended): Loops, including branch instructions, that have not been the target of optimization level-2 loop unrolling, are the target of this extended loop unrolling.
  - Optimization function repeated execution: In optimization function repeated execution, the optimization features except the loop unrolling feature will be repeatedly executed until no more optimization is needed. However, the translation time will increase.
- 4(Optimization level 4): The following optimization features are exercised in addition to the features provided by optimization level 3.
  - Arithmetic operation evaluation type change (same as effected by -K EOPT specifying): Performs optimization to change arithmetic operation evaluation type at compilation stage. When this option is specified, there may be side effects on the execution results.
  - Standard function expansion/change (same as effected by -K LIB specifying): Switches to a higher-speed standard function that recognizes standard function operations, performs standard function inline expansion, and performs identical operations. When this option is specified, there may be side effects on the execution results. Since standard function inline expansion is implemented, the code size may increase.

● -K EOPT option and -K NOEOPT option

The EOPT suboption effects optimization by changing the arithmetic operation evaluation type at the compilation stage. When this option is specified, side effects may occasionally be produced on the execution results. This option takes effect only when an optimization level of 1 or higher is specified by the -O option. The NOEOPT suboption cancels the EOPT suboption.

[Example]

```
Input;    extern int i;
          void func(int a, int b){
            i=a-100+b+100;
            }
Operation: fcc911s -K eopt -0 - cpu MB91F154 -S sample.c
Output:           ADD     R5, R4    ; Order of arithmetic operation replaced
                  LDI;32  #_i, R12
                  ST      R4, @R12
```

● -K LIB option and -K NOLIB option

The LIB suboption recognizes the standard function operation and replaces the standard function with a higher-speed standard function which effects standard function in-line expansion and performs the same operation as the original standard function. When this option is specified, side effects may be produced on the execution results. Since standard function inline expansion is implemented, the code size may increase.

This option takes effect only when an optimization level of 1 or higher is specified by the -O option.

The NOLIB suboption cancels the LIB suboption.

[Example]

```
Input:     extern int i;
           void func(void) {
             i=strlen("ABC");
              }
Operation: fcc911s -K lib -O -cpu MB91F154 -S sample.c
Output:           LDI    #3,R0   ; Processing equivalent to strlen expanded
                  LDI:32 #_i, R12
                  ST     R0, @R12
```

● -K {LONGADDRESS [= {CODE|DATA}] | SHORTADDRESS [= {CODE|DATA}]}option

The SHORTADDRESS suboption generates code on the presumption that the (address) of a symbol to be loaded within the program is within the 20-bit expression range.

When CODE or DATA is specified, only the symbols of the section allocated in the ROM (CODE or CONST) or the symbols of the section allocated in the RAM (DATA or INIT) are to be processed. If the address exceeds the range that can be expressed by 20 bits, an error occurs during linking. Operation is normal even if symbols other than those loaded in the program are positioned outside the range of the addresses that can be expressed by 20 bits. The LONGADDRESS suboption enables handling symbol addresses as 32-bit addresses. If the SHORTADDRESS suboption or LONGADDRESS suboption is omitted, -K LONGADDRESS is applied.

[Example]

```
Input:     extern int i;
           extern void sub(void);
           void func(void) {
             i=10;
             sub( );
           }
Operation: fcc911s -K shortaddress -O -S -cpu MB91F154 sample.c
Output:            LDI:20   #_i, R12     ; 20-bit symbol used
                   LDI      #10, R0
                   ST       R0, @R12
                   CALL20   _sub, R12     ; 20-bit symbol used
```

● -K CNC option and -K NOCNC option

CNC suboption specifies the symbol (address) of CONST section to contrary of CODE section specified by -K LONGADDRESS option or -K SHORTADDRESS option. As a result, the symbol size of the CODE section, the CONST section, and the DATA section (included INIT section) can be individually set. Table 3.5-3 shows the symbol size when options are specified.

The NOCNC suboption cancels the CNC suboption.

Table 3.5-3 The symbol bit size when options are specified

Specified Option	-K longaddress=CODE		-K shortaddress=CODE	
	-K CNC	-K NOCNC	-K CNC	-K NOCNC
The symbol size for the CODE section	32 bits	32 bits	20 bits	20 bits
The symbol size for the CONST section	20 bits	32 bits	32 bits	20 bits

● -K NOALIAS option and -K ALIAS option

The NOALIAS suboption optimizes the data specified by the pointer on the assumption that the pointer does not specify the same area of different pointers.

These options are enabled only when level 1 or more optimization is specified in the -O option. The language specification permits different pointers to include the same area. Therefore, when using this option, check the program carefully.

The ALIAS suboption cancels the NOALIAS suboption.

[Example]

```
Input:     extern int i;
           extern int j;
           void func9(int *p) {
             *p=i+1;
             j=i+1;
           }
Operation: fcc911s -K noalias -O -cpu MB91F154 -S sample.c
Output:           LDI:32  #_i, R12
                  LD      @R12, R0
                  LDI:32  #_j, R12
                  ADD     #1, R0
                  ST      R0, @4
                  ST      R0, @12      ; Value of *p=i+1 reused
```

● -K {SCHEDULE|NOSCHEDULE} option

-K {SCHEDULE|NOSCHEDULE} option specifies whether or not to implement instruction scheduling. When the SCHEDULE suboption is specified for the fcc911s command, instruction scheduling will be conducted. When the NOSCHEDULE suboption is specified, the command will not conduct instruction scheduling. If this option is omitted, the operation conforms to the contents specified in the -O option. When an optimization level of 1 or higher is specified by the -O option, -K SCHEDULE is assumed to be specified.

The SCHEDULE suboption cancels the NOSCHEDULE suboption.

● -K NOUNROLL option and -K UNROLL option

The NOUNROLL suboption inhibits loop unrolling optimization. Use this option when loop unrolling optimization is to be inhibited with the -O2 to -O4 options specified. The UNROLL suboption cancels the NOUNROLL suboption. When -O0 or O1 is specified, these options are invalid because loop unrolling is not optimized.

- -x function name 1 [, function name 2,...]option and -Xx option

  The -x option effects in-line expansion, instead of function calling, of functions defined by a C source. However, recursively called functions will not be subjected to in-line expansion. It should also be noted that functions may not be subjected to in-line expansion depending on asm statement use, setjmp function calling, and other conditions. The -x option takes effect only when level 1 or more optimization is specified simultaneously with the -O option.

  The -Xx option cancels the -x option.

  [Example]

```
Input:     extern int a;
           static void sub(void) { a=1; }
           void func(void) { sub( ); }
Operation: fcc911s -cpu MB91F154 -O -x sub -S sample.c
Output:    _func:
                   LDI     #1, R0
                   LDI:32  #_a, R12
                   RET:D
                   ST      R0, @R12
```

- -xauto [size]option and -Xxauto option

  The -xauto option effects in-line expansion, instead of function calling, of functions whose logical line count is not less than size. However, recursively called functions will not be subjected to in-line expansion. It should also be noted that functions may not be subjected to in-line expansion depending on asm statement use, setjmp function calling, and other conditions. If the size entry is omitted, the value "30" is assumed to be specified. The values from 1 to 127 are valid for size. The -xauto option takes effect only when level 1 or more optimization is specified simultaneously with the -O option.

  The -Xxauto option cancels the -xauto option.

  [Example]

```
Input:     extern int a;
           static void sub(void){ a=1; }
           void func(void) { sub( ); }
Operation: fcc911s -cpu MB91F154 -O -xauto -S sample.c
Output     _func:
                   LDI    #1, R0
                   LDI:32 #_a, R12
                   RET:D
                   ST     R0,@R12
```

● -K MERGESTRING option and -K NOMERGESTRING option

The MERGESTRING suboption merges the substance of the same character string literal.

The NOMERGESTRING suboption cancels MERGESTRING suboption.

[Example]

```
Input:  char *a = "abcdef";
        char *b = "abcdef";
Operation: fcc911s -cpu MB91101 -S sample.c -K SPEED -K MERGESTRING
Output:    .section    INIT, DATA, align=4
           .global _b
           .align  4
       _b:
           .word  LS_0
           .global _a
       _a:
           .word  LS_0
           .section     CONST, CONST, align=4
           .align  4
       LS_0:
           .ascii "abcdef\000"
           .datab.b     1,0
```

● -K NOCALL21 option and -K CALL21 option (For FR81 only.)

The NOCALL21 suboption specifies not to use the CALL21 instruction. The CALL21 suboption cancels the NOCALL21 suboption.

The default is -K CALL21.

Caution:

- The compiler outputs the CALL21 instruction when FR81 is used and the -K NOCALL21 is not specified. If -K NOCALL21 is specified, the compilerÅfs behavior depends on -K {LONGADDRESS| SHORTADDRESS}=CODE.
- These options are enabled when FR81 is specified by the -cpu option, otherwise these are ignored.

● -K BP option and -K NOBP option (For FR81 only.)

The BP suboption specifies generating Base Pointer register (BP) relative access instructions, so that the access by an instruction to the variable which is in the fixed range from the address of BP becomes possible. Please see section "6.2 Startup Routine Creation" for the detail of setting the address of BP.

The distance between the address of BP and the variable is not clarified until linking. Therefore, the liker error (E4329L: Value out of range (value)) occurs when the variable out of the range. In this case, please do not specify the BP suboption.

The NOBP suboption cancels the BP suboption.

The default is -K NOBP.

Caution:

- Please do not specify -K BP when libraries or objects which may use other projects are compiled. When variables output by the BP relative access are placed out of the range of the access, recompile is needed.
- The compiler generates BP relative access instructions when FR81 is used and the -K BP is specified. If -K NOBP is specified or nothing is specified, the compiler's behavior depends on -K {LONGADDRESS|SHORTADDRESS}=DATA.
- The range of the BP relative access depends on boundary alignment. Moreover, it does not depend on -K A4. Please see section "4.3 fcc911s Command Boundary Alignment" for the detail of boundary alignment.
  - 1-byte-boundary alignment : from BP to BP + 64KB
  - 2-byte-boundary alignment : from BP to BP + 128KB
  - 4-byte-boundary alignment : from BP to BP + 256KB
- These options are enabled when FR81 is specified by the -cpu option, otherwise these are ignored.

[Remarks]

BP relative access instructions are 8 instructions as follows:

- LD @(BP, udisp18), Ri
- LDUH @(BP, udisp17), Ri
- LDUB @(BP, udisp16), Ri
- FLD @(BP, udisp18), Ri
- ST Ri, @(BP, udisp18)
- STH Ri, @(BP, udisp17)
- STB Ri, @(BP, udisp16)
- FST Ri, @(BP, udisp18)

Please refer to the hardware manual for the detail of these instructions,

● -K NOSAVEFREG option and -K SAVEFREG option (For FR81 with FPU only.)

The NOSAVEFREG suboption specifies not to generate save/restore code of FPU registers (from FR0 to FR15 and FCR) in an interrupt handler, so that the code size is reduced. However, the data of FPU registers might be destroyed.

When there is no guarantee that an interrupt handler does not destroy the data of FPU registers, either do not use NOSAVEFREG suboption or insert the save/restore code of FPU registers in an interrupt handler.

The SAVEFREG suboption cancels the NOSAVEFREG suboption.

The default is -K SAVEFREG.

Caution:

These options are enabled when FR81 is specified by the -cpu option, otherwise these are ignored.

3.5.6 Output Object Related Options

This section describes the options related to output object formats.

■ Output Object Related Options

● -align {FUNC4|FUNC8} option and -Xalign option

The -align option aligns branch labels.

When the FUNC4 suboption is specified, the compiler suffixes CODE section names with _4, and selects 4 as the section boundary value. When the FUNC8 suboption is specified, the compiler suffixes CODE section names with _8, and selects 8 as the section boundary value.

When the -s option or #pragma section is used to specify a section arrangement address, it overrides the -align option.

The FUNC4 suboption aligns the beginnings of functions on 4-byte boundaries, while the FUNC8 suboption aligns them on 8-byte boundaries.

The -Xalign option disables the align value specified for labels. When both -align and -Xalign options are omitted, the -Xalign option is used.

[Remarks]

With some CPU of the FR, instructions are fetched on 8-byte boundaries, in units of 4/8 bytes. If the branch destination address is not 4/8-byte boundaries when a branch occurs, extra fetch cycles for 1 to 3 cycles are generated.

The -align option can be used to avoid this extra fetch cycle.

Note that branch labels aligned on boundaries increase the code size.

[Example]

```
Input:     void foo(void){}
Operation: fcc911s  -cpu  MB91F154  -O  -S  sample.c -align func8
Output:    .SECTION CODE_8, CODE, ALIGN=8 ;The section name is changed.
           .ALIGN  8                   ;;8-byte boundary alignment.
           .GLOBAL _foo
       _foo:
           RET
```

● -cpu MB number option

Specifies MB number of CPU to be used. This option cannot be omitted.

[Example]

```
>fcc911s -S -cpu MB91F154 sample.c
```

● -cif filename option

The -cif option specifies the CPU information file name for filename. The CPU information describes the information about the CPU that has the MB number specified in the -cpu option.

[Example]

```
>fcc911s -S -cpu MB91F154 -cif lib\911\911.csv sample.c
```

● -s defname = newname [, attr [, address]] option and -Xs option

Changes the compiler output section name from defname to newname, and changes section type to attr. An arrangement address can be specified in the address option.

For compiler output section names, see section "4.1 Section Structure of fcc911s Command". For selectable section types, refer to the Assembler Manual.

If the arrangement address is specified, the arrangement address cannot be specified relative to the associated section at linking.

The -Xs option cancels the -s option.

[Example]

```
Input:   void func(void) { }
Operation: fcc911s -s CODE=PROGRAM, CODE, 0x1000 -S -cpu MB91F154 sample.c
Output          .SECTION PROGRAM, CODE, LOCATE=H'00001000
         ;-------begin_of_function
                 .GLOBAL _func
         _func:
              ST    RP, @-SP
              ENTER #4
         L_main:
              LEAVE
              LD    @SP+, RP
              RET
```

● -K {A1|A4} option

Specifies the minimum allocation boundary for external and static variables. The A4 suboption selects the 4-byte minimum allocation boundary.

When the 4-byte minimum allocation boundary is used, more efficient code may be generated through inline expansion of a character string manipulation function when -K lib is specified. This code malfunctions if boundary alignment is incorrect. Therefore, objects for which different allocation boundaries are specified must not be linked. If such objects are linked, unnecessary areas are generated as a result of consistent boundary alignment, causing an increase in the number of objects. The A1 suboption selects the 1-byte minimum allocation boundary.

When this option is omitted, -K A1 is used.

[Example]

```
Input:     char c1, c2;
Operation: fcc911s -K A4 -S -cpu MB91F154 sample.c
Output:            .SECTION   DATA, DATA, ALIGN=4
                   .GLOBAL _c2
           _c2:    .RES.B 4   ; Positioned at 4-byte boundary
                   .GLOBAL _c1
           _c1:    .RES.B 4   ; Positioned at 4-byte boundary
```

● -varorder {sort|normal} option

The -varorder normal option causes static variables to be stored in memory in the order in which they are described in the source. Variables with an initial value and those without initial value are stored in different sections, so description with these sections mixed prevents the variables from being arranged in the order in which they are described in the source.

The -varorder sort option sorts the order of storing static variables in memory according to the order of variable alignment. Specifying this option reduces the size of unused static variable areas.

[Example]

```
Input:           long a;
                 char b;
                 long c;
                 char d;
Operation: fcc911s -S -varorder normal -cpu MB91F154 sample.c
Output:
                 .ALIGN 4
                 .GLOBAL _a
           _a:
                 .RES.B 4
                 .ALIGN 1
                 .GLOBAL _b
           _b:
                 .RES.B 1
                 .ALIGN 4
                 .GLOBAL _c
           _c:
                 .RES.B 4
                 .ALIGN 1
                 .GLOBAL _d
           _d:
                 .RES.B 1
```

- -CO option and -XCO option

  The -CO option generates the compatible object. When the target CPU is FR or FR80, the option generates the compatible object for all FR architecture. This object can link to all FR objects. When the target CPU is FR81 with or without FPU, the option generates the compatible object for FR81 architecture. This object can link to FR81 with or without FPU objects. When the -CO option is not specified, the object of the executable format for the architecture specified by the -cpu option is generated.

  The -XCO option cancels the -CO option.

- -t {none|used|all||local} option

  This option is valid only for the C++ source. It specifies the type of template instantiation.

  When the -t none option is specified with the --no-auto_instantiation option, any template instantiation is not generated. When this option is specified with the --auto_instantiation option, all instantiation is automatically generated.

  The -t used option instantiates only the parameter-type template functions and template member functions used in the module.

  The -t all option instantiates even unused member functions if they are of parameter type and template class used in the module.

  The -t local option generates template functions and template member functions used in the module as the in-module local functions when the -t local is specified, --auto_instantiation option is invalid.

  Note that overlapping instantiation by the used and all options is not allowed when more than one module is linked.

  For information on the template, see section "9.6 Limitations on Use of C++ Template".

  The default is -t none.

- --no_auto_instantiation option and --auto_instantiation option

  These options are valid only for the C++ source.

  When the --auto_instantiation option is specified, the template instantiation is automatically done.

  The --no_auto_instantiation option provides manual template instantiation. Instantiation must be controlled by the -t {used |all| local} option, #pragma, or declaration explicitly.

  For information on using #pragma to control instantiation, see section "5.11 Function for Controlling Instantiation of C++ Template".

- --suppress_vtbl option and --force_vtbl option

  These options are valid only for the C++ source. They change the generation type of a virtual function table.

  The --suppress_vtbl option causes the virtual function table to be referenced from other modules without being generated in a module.

  The --force_vtbl option causes the virtual function table to be generated in a module for reference from other modules.

  These options are valid only when the member functions specified as being virtual are in the class and all of them are defined inline. If these options are not given under this condition, a virtual function table is generated locally for each module.

[Example] Cases for option

```
class FOO {
   int a;
public:
   FOO( ) {a=0;}
   virtual void memfunc(int x) {a=x;}
}; //All the member functions are defined inline.
```

The --force_vtbl option must be specified for at least one module. Note that overlapping table definition is not allowed when specifying the option for more than one module.

● -K NOFPU option and -K FPU option (For FR81 with FPU only.)

These options are enabled when specifying FR81 with FPU by the -cpu option, otherwise these are ignored.

The NOFPU suboption specifies not to use FPU instructions.

The FPU suboption cancels the NOFPU suboption.

The default is -K FPU.

Caution:

- TWhen -K NOFPU is specified, -K {NOSAVEFREG | SAVEFREG} are ignored. Therefore, FPU registers (from FR0 to FR15 and FCR) are not saved.
- These options are enabled when FR81 is specified by the -cpu option, otherwise these are ignored.

3.5.7 Debug Information Related Options

This section describes the options related to the debug information to be referenced by the symbolic debugger.

■ Debug Information Related Options

● -g option and -Xg option

-g option adds debugging information to the object file. The -Xg option cancels -g option.

When optimization options are specified, please note the following issues.

- The breakpoint might not be able to be set.
  Because the line might be moved or deleted, the breakpoint might not be able to be set.

- Plural breakpoints might be set at once.
  Because the lines from which the instructions are deleted might be consecutive, plural breakpoints might be set at once.

- It might not stop at the breakpoint.
  Because the line might be moved or deleted, it might not stop at the breakpoint.

- The value of the variable displayed in the watch window might not be correct.
  Because the instruction that stores the value of the variable might be moved, the timing to be updated might not correspond to the order of the C source.
  Because one register might be assigned to two or more variables, the timing to be updated might not correspond to the order of the C source.

- The local variable and the parameter might not be able to be referred to by the debugger.
  The optimized local variable and parameter might not be able to be referred to by the debugger.

- The CALL STACK function and the STEP OUT function might not be able to be used by the debugger.
  When the prologue/epilogue code of the function is optimized, the CALL STACK function (SHOW CALLS command) and the STEP OUT function (GO /RETURN command) cannot be used by the debugger.

- When the function inlining (-x func or -xauto option) is specified, the C source displayed in the source window might not be correct.

3.5.8 Command Related Options

This section describes the options related to the other tools called by the command.

■ Command Related Options

● -Y item, dir option and -XY option

Changes the item position to the dir directory. The -XY option cancels the -Y option. The item is one of the following.

- p: Left for compatibility with the previous version but does not provide anything.
- c: Changes the compiler pathname to dir.
- a: Changes the assembler pathname to dir.
- l: Changes the linker pathname to dir.

[Example]

```
>fcc911s file.c  -Y c, \home\newlib -cpu MB91F154
```

Calls the compiler as \home\newlib\cpcoms.

● -T item, arg1 [, arg2...]option and -XT option

Passes arg to item as an individual compiler tool argument. The -XT option cancels the -T option.

Use a comma to separate arguments. To describe a comma as an argument, position a backslash (\) immediately before the comma. The comma positioned after the backslash will not be interpreted as a delimiter. To write a blank as an argument, describe a comma in place of a blank.

For the options for various commands, refer to the associated manuals. The following can be specified as the item.

- a: Assembler
- l: Linker

[Example]

```
>fcc911s -T a, -lf, asmlist file.c -cpu MB91F154
```

Sequentially passes arguments "-lf " and "asmlist" to the assembler. Therefore, the assemble list asmlist will be generated as a result of command execution.

3.5.9 Linkage Related Options

The linkage related options are related to linkage.

■ Linkage Related Options

- -e name option and -Xe option

  The -e option sets the entry symbol to name at linking. The -Xe option cancels the -e option. Since the option definition is usually provided in the startup routine, this option does not have to be specified.

  For details of the option, refer to the Linkage Kit Manual.

- -L path1 [, path2...] option and -XL option

  The -L option adds path to the library path used at linking to search for a library to be linked. If the option is not specified, ${LIB911} is selected in the fcc911s command automatically.

  The -XL option cancels the -L option.

  For details of the option, refer to the Linkage Kit Manual.

- -l lib 1 [, lib 2...] option and -Xl option

  The -l option specifies the name (lib) of the library to be linked at linking.

  If the extension entry is omitted, the .lib extension is added automatically.

  The -Xl option cancels the -l option.

  For the objects output by the compiler, by default, in fcc911s command lib911.lib is set as the names of the libraries to be linked.

  For the details of the option, refer to the Linkage Kit Manual.

- -m option and -Xm option

  The -m option generates a map file at linking.

  A map file output with a file name with the .map extension is generated in the current directory.

  The -Xm option cancels the -m option.

- -ra name = start/end option and -Xra option

  The -ra option specifies the RAM area at linking. The -Xra option cancels the -ra option. For details of the option, refer to the Linkage Kit Manual.

- -ro name = start/end option and -Xro option

  The -ro option specifies the ROM area at linking. The -Xro option cancels the -ro option. For details of the option, refer to the Linkage Kit Manual.

● -sc param option and -Xsc option

The -sc option specifies the section arrangement at linking.

The -Xsc option cancels the -sc option.

If the option is not specified, -sc IOPORT=0,*=0x1000 is selected automatically.

For details of the option, refer to the Linkage Kit Manual.

● -startup filename option and -Xstartup option

The -startup option selects filename as the object file name of the startup routine to be linked at linking.

If this -startup option is not specified, %FETOOL%\lib\911\startup.obj is selected in the fcc911s command automatically.

The -Xstartup option cancels the -startup option.

For details of the startup routine, see "CHAPTER 6 EXECUTION ENVIRONMENT".

3.5.10 Option File Related Options

This section describes the option file related options.

■ Option File Related Options

● -f filename option and -Xf option

-f option used to read the specified option file (filename). If the option file name does not have an extension, an .opt extension will be added.

The command options can be written in an option file. For the details of an option file, see section "3.6 Option Files".

The -Xf option cancels the -f option.

● -Xdof option

-Xdof option specifies that the default option file will not be read.

For the default option file, see section "3.6 Option Files".

3.6 Option Files

This section explains option files for commands. By writing options in a file, a group of options can be specified. This feature also permits you to put startup options to be specified in a file.

■ Option File

Option file reading takes place when an associated option is specified. This assures that the same result is obtained as when an option is specified at the -f specifying position in the command line.

If the option file name is without an extension, an .opt extension will be added.

● Option File General Format

All entries that can be made in a command line can be written in an option file.

A line feed in an option file is replaced by a blank.

A comment in an option file is replaced by a blank.

[Example]

```
-I /usr/include   # Include specifying
-D FR             # Macro specifying
-g                # Debug data generation specifying
-S                # Execution of processes up to compiling
```

■ Option File Limitations

The -f option can be written in an option file. However, nesting is limited to 8 levels.

■ Acceptable Comment Entry in Option File

A comment can be started from any column.

A comment is to begin with a sharp (#). The entire remaining portion of the line serves as the comment.

In addition, the following comments can also be used.

- /* Comment */
- // Comment
- ; Comment

[Example]

```
-I /usr/include   #  Include specifying
-D FR             /* Macro specifying */
-g                // Debug data generation specifying
-S                ;  Execution of processes up to compiling
```

■ Default Option File

A preselected option file can be read to initiate command execution. The obtained result will be the same as when an option is specified prior to another option specified in the command line.

The default option file name is predetermined as follows.

[For UNIX OS]

${OPT911}/fcc911.opt

[For Windows]

%OPT911%\fcc911.opt

The fcc911s command default option file name is fcc911.opt.

If the default option file does not exist in the specified directory, such a specifying is ignored.

To inhibit the default option file feature, specify the -Xdof option in the command line.

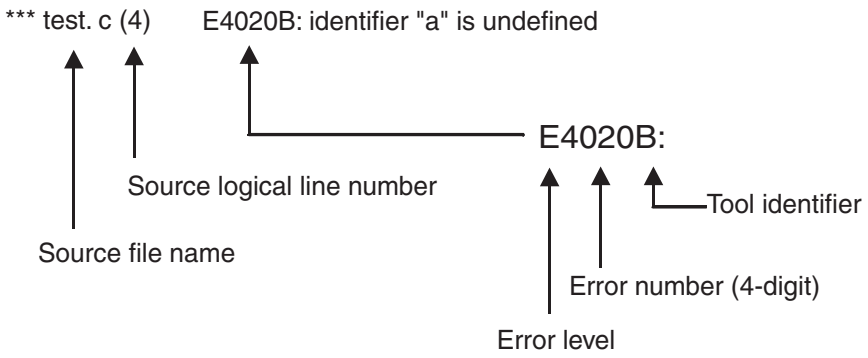
3.7 Messages Generated in Translation Process

When an error is found in a source program or a condition which does not constitute a substantial error but requires attention is encountered, diagnostic messages may be generated at the time of translation. For message outputs generated by tools other than the compiler, refer to the respective manuals for the tool.

■ Messages Generated in Translation Process

A diagnostic message output example is shown in Figure 3.7-1.

Figure 3.7-1 Diagnostic Message Example



■ Tool Identifier

The tool identifier indicates the tool that has detected the error.

- D: Command
- B: Compiler
- S: Scheduler (Internal tool for compiler)
- A: Assembler
- L: Linker

■ Error Level

The error level represents the diagnostic check result type.

Table 3.7-1 describes the relationship between various error levels and return codes and their meanings.

Table 3.7-1 Relationship between Error Levels and Return Codes

Error Level	Return Code	Meaning
I	0	Indicates a condition which does not constitute an error but requires attention.
W	0	Indicates a minor error. Process execution continues without being interrupted. The return code can be changed by the -cwno option.
E	2	Indicates a serious error. Process execution stops.
F	3	Indicates a fatal error which is related to quantitative limitations or system failure. Process execution stops.

CHAPTER 4

fcc911s COMMAND OBJECT PROGRAM STRUCTURE

This chapter explains about the information necessary for program execution.

4.1 Section Structure of fcc911s Command

The fcc911s command has the following seven sections:

- **Code section**
- **Initialized section**
- **Constant section**
- **Data section**
- **I/O section**
- **Vector section**
- **C++ Init section**

■ fcc911s Command Section Structure

Table 4.1-1 shows the sections to be generated by the compiler and their meanings.

Table 4.1-1 fcc911s Command Section List

No.	Section Type	Section Name	Type	Boundary Alignment [Byte]	Write	Initial Value
1	Code section	CODE	CODE	2	Disabled	Provided
2	Initialized section	INIT	DATA	4	Enabled	Provided
3	Constant section	CONST	CONST	4	Disabled	Provided
4	Data section	DATA	DATA	4	Enabled	Not provided
5	I/O section	IO	IO	4	Enabled	Not provided
6	Vector section	INTVECT	CONST	4	Disabled	Provided
7	C++Init section	EXT_CTOR_DTOR	CONST	4	Disabled	Provided

The purpose of each section use and the relationship to the C/C++ language are explained below.

● Code section

Code section stores machine codes. This section corresponds to the procedure section for the C language.

● Initialized section

Initialized section stores the initial value attached variable area. For the C language, this section corresponds to the area for external variables without the const attribute, static external variables, and static internal variables.

- Constant section

  Constant section stores the write-protected initial value attached variable area. For the C language, this section corresponds to the area for const attribute attached external variables, static external variables, and static internal variables.

- Data section

  Data section stores the area for variables without the initial value. For the C language, this section corresponds to the area for external variables (including those which are with the const attribute), static external variables, and static internal variables.

- I/O section

  I/O section stores the area for the __io-qualified variables. For the C language, this section corresponds to the area for __io-qualified external variables (including those which are provided with the const attribute), static external variables, and static internal variables.

  The default section name is IO.

- Vector section

  This section stores the interrupt vector tables. In the C language, a vector table is generated only when its generation is specified in #pragma intvect. The default section name is INTVECT.

- C++ Init section

  This section stores tables for indicating the entry of functions constituting and destroying static class objects. It must be used at start-up. For information on specifying the startup program, see section "6.2 Startup Routine Creation".

4.2 Rules for Name Generation with the fcc911s

The rules for the names used by the fcc911s are explained below.

■ Rules for Name Generation with the fcc911s

Table 4.2-1 shows the relationship between the names generated by the compiler and the C language.

Table 4.2-1 Label Generation Rules

C Language Counterpart	Label Generated by Compiler
Function name	-function name
External variable name	-external variable name
Static variable name	LI_no
Local variable name	-
Virtual argument name	-
Character string, derived type	LS_no
Automatic variable initial value	LS_no
Target location label	L_no

Note: The compiler internal generation number is placed at the no position.

4.3 fcc911s Command Boundary Alignment

The standard data type and boundary alignment are explained below. Table 4.3-1 shows the assignment rules.

■ fcc911s Command Boundary Alignment

Table 4.3-1 fcc911s Command Variable Assignment Rules

Variable Type	Assignment Size [Byte]	Boundary Alignment [Byte]
_Bool	1	1
char	1	1
signed char	1	1
unsigned char	1	1
short	2	2
unsigned short	2	2
int	4	4
unsigned int	4	4
long	4	4
unsigned long	4	4
float	4	4
double	8	4
long double	8	4
pointer/address	4	4
Structure/union/class	See "4.5 fcc911s Command Structure/Union"	

Note: Some variables are aligned on 4-byte boundaries when the -K A4 option is specified. The -K A4 option does not affect structure/union member boundary alignment.

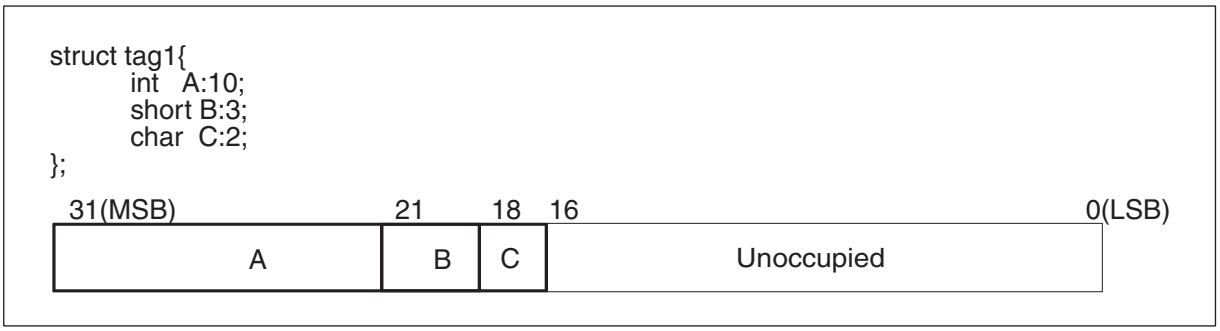
4.4 fcc911s Command Bit Field

The fcc911s command bit field data size and boundary alignment are explained below. The bit field data is assigned to a storage unit that has an adequate size for bit field data retention and is located at the smallest address.

■ fcc911s Command Bit Field

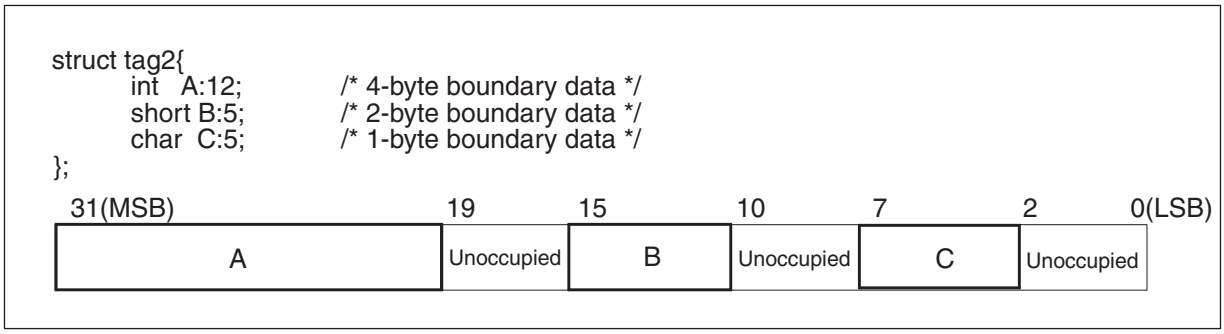
Consecutive bit field data are packed at consecutive bits having the same storage unit, without regard to the type, beginning with the LSB and continuing toward the MSB. An example is shown in Figure 4.4-1.

Figure 4.4-1 fcc911s Command Bit Field Data Size and Boundary Alignment Example 1



If a field to be assigned lies over a bit field type boundary, its assignment is completed by aligning it with a boundary suitable for the type. An example is shown in Figure 4.4-2.

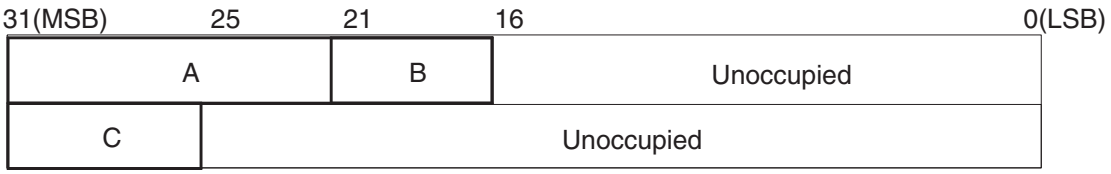
Figure 4.4-2 fcc911s Command Bit Field Data Size and Boundary Alignment Example 2



When a bit field having a bit length of 0 is declared, it is forcibly assigned to the next storage unit. An example is shown in Figure 4.4-3.

Figure 4.4-3 fcc911s Command Bit Field Data Size and Boundary Alignment Example 3

```
struct tag3{
        int   A:10;
        int   B:5;
        int    :0;
        int   C:6;
};
```



4.5 fcc911s Command Structure/Union

The structure/union of fcc911s command data size and boundary alignment are explained below. The structure/union data size is a multiple of the maximum boundary alignment size of the members. Boundary alignment for the area itself is accomplished by means of member maximum boundary alignment. The individual members are subjected to boundary alignment in accordance with the member type.

■ fcc911s Command Structure/Union

Figure 4.5-1 to Figure 4.5-3 show examples concerning structure/union data size and boundary alignment.

Figure 4.5-1 fcc911s Command Structure/Union Data Size and Boundary Alignment Example 1

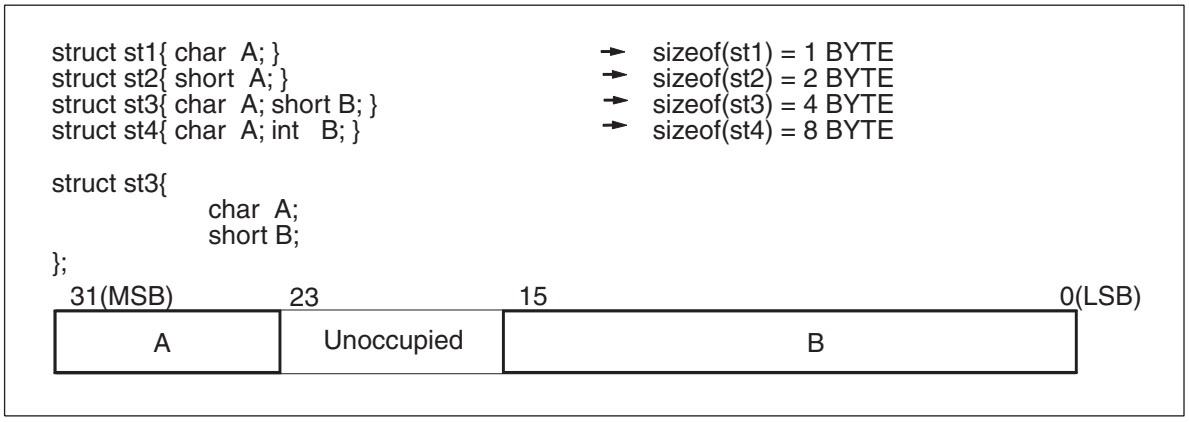


Figure 4.5-2 fcc911s Command Structure/Union Data Size and Boundary Alignment Example 2

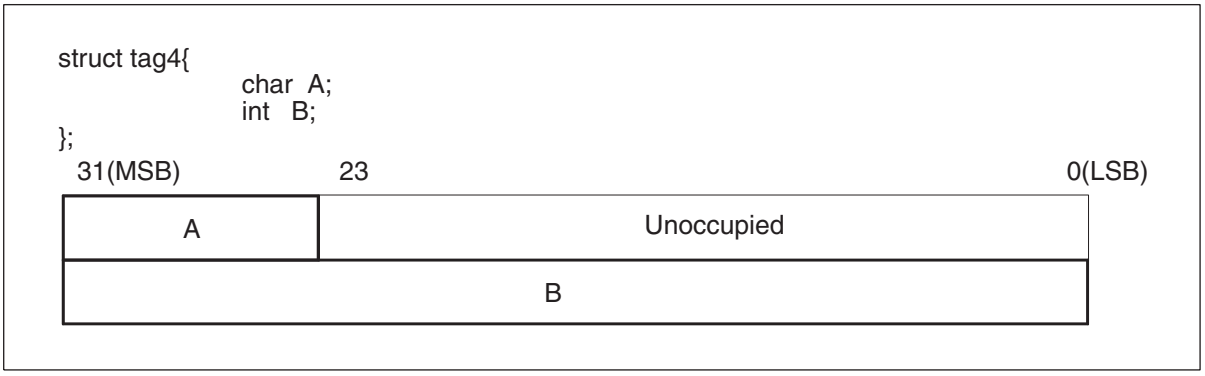
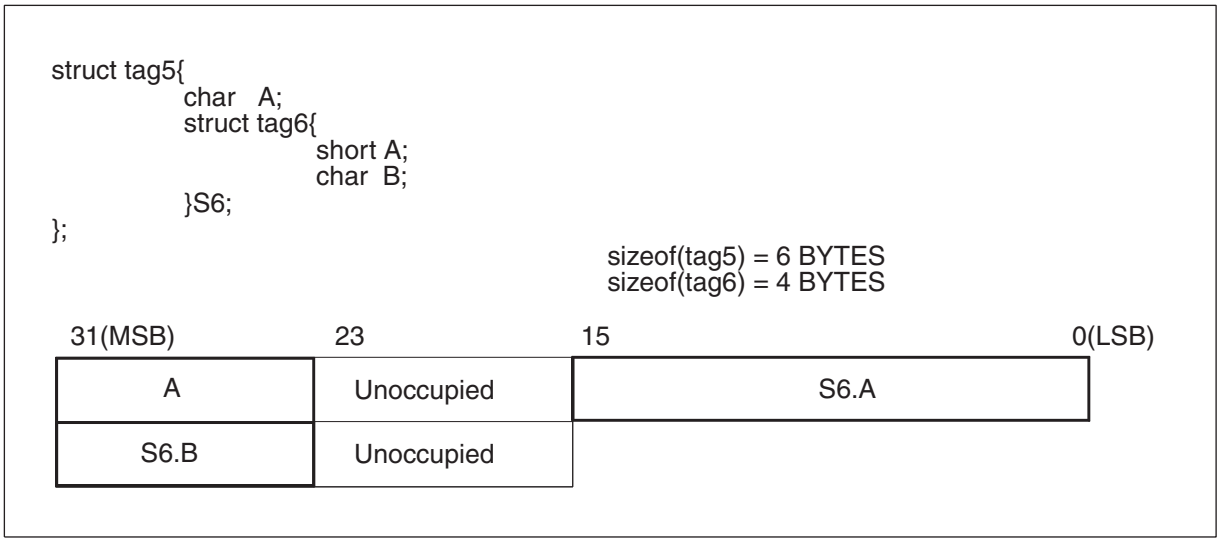


Figure 4.5-3 fcc911s Command Structure/Union Data Size and Boundary Alignment Example 3



4.6 fcc911s Command Function Call Interface

The general rules for control transfer between functions are established as standard regulations for individual architectures and are called standard linkage regulations. A module written in C language can be combined with a module written using a different method (e.g., assembler language) when the standard linkage regulations are complied with.

■ fcc911s Command Function Call Interface

- Stack frame

  The stack frame construction is stipulated by the standard linkage regulations.

- Argument

  Argument transfer relative to the callee function is effected via a stack or register.

- Argument extension format

  When an argument is to be stored in a stack, the argument type is converted to an extended format in accordance with the argument type.

- Calling procedure

  The caller function initiates branching to the callee function after argument storage.

- Register

  The register guarantee stated in the standard linkage regulations and the register setup regulations are explained later.

- Return value

  The return value interface stated in the standard linkage regulations is explained later.

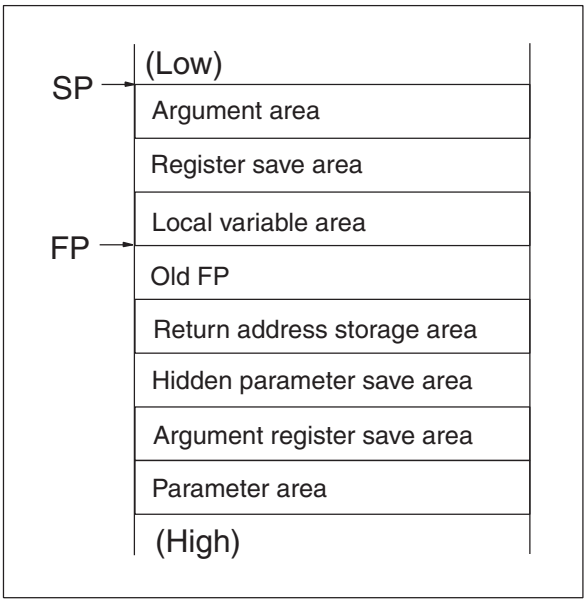
4.6.1 fcc911s Command Stack Frame

The standard linkage regulations prescribe the stack frame construction.

■ fcc911s Command Stack Frame

The stack pointer (SP) always indicates the lowest order of the stack frame. Its address value always represents the word boundary. Figure 4.6-1 show the standard function stack frame status.

Figure 4.6-1 fcc911s Command Stack Frame



● Argument area/Parameter area

When a function is called, this area is used for argument transfer. This area is referred to an argument area when the caller function is used to set up an argument and to a parameter area when the argument is referenced by the callee function. This area is created when all arguments cannot be contained in an argument register during the transfer of an argument.

For details, see section "4.6.2 fcc911s Command Argument".

● Register save area

This is a register save area that must be guaranteed for the caller function. This area is not secured when the register save operation is not needed.

● Local variable area

This is the area for local variables and temporary variables.

● Old FP

This area stores the frame pointer (FP) value of the caller function.

- Return address storage area

  This area saves the RP. The RP stores the address of a return to the caller function for the purpose of function calling.

- Hidden parameter save area

  This area stores the start address of the return value storage area for a structure/union return function.

  When a structure/union is used as the return value, the caller function stores the return value storage area start address in register R4 and passes it to the callee function.

  The callee function interprets the address stored in the R4 as the return value storage area start address.

  When register R4 needs to be saved into memory, the callee function saves it in the hidden parameter save area. This area is not secured when the save operation is not needed.

- Argument register save area

  This area saves the argument register. This area is not secured when the save operation is not needed.

  For details, see section "4.6.2 fcc911s Command Argument".

4.6.2 fcc911s Command Argument

Arguments, the count of which equals the count of argument registers (4 words), are positioned in registers R4 to R7 and delivered to the callee function. When a structure/union return function is called, three argument registers (R5 to R7) are used because the return value area address is stored in register R4. Arguments not placed in the argument registers will be stored in the stack actual argument area for transfer purposes. When an 8-byte type argument is to be delivered using registers, it is divided into two and placed in two registers for transfer.

■ fcc911s Command Argument

When argument registers must be saved to memory, the callee function secures an argument register save area in the stack. In this case, a continuous argument register save area must be established in the parameter area. The argument register save area must be allocated as needed to cover the size of the argument register to be saved.

If the function has a variable count of arguments, it saves all argument registers in the argument register save area.

Caution:

In a C++ program, arguments that do not appear in the source program may be passed. The order and location in which arguments are stacked may or may not be as desired.

[Example 1]

```
double d;
sub(d); → The high-order words of d are delivered by R4,
          and the low-order words of d are delivered by R5.
```

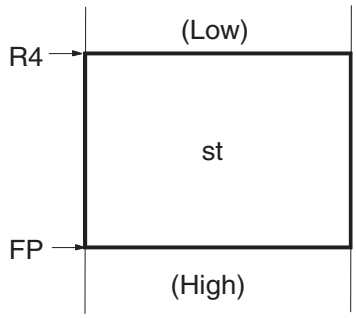
[Example 2]

```
int a, b, c;
double d;
sub(a, b, c, d); → a is delivered by R4, b by R5, and c by R6.
                   The high-order words of d are delivered by R7,
                   and the low-order words of d are delivered by
                   the stack.
```

When a structure/union is to be delivered as an argument, the caller copies the structure to the local variable area and passes the address of that area to the callee. In this case, if the structure/union size is less than 4 bytes or is not divisible by 4, the less-than-4-byte fraction is handled as one 4-byte unit.

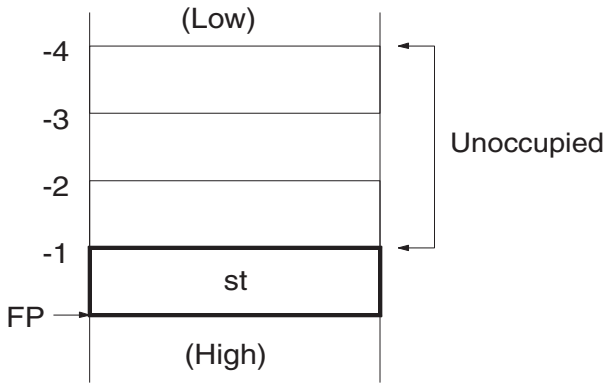
[Example 3]

```
struct A st;
sub(st);
```



[Example 4]

```
struct A{ char a; }st;
```

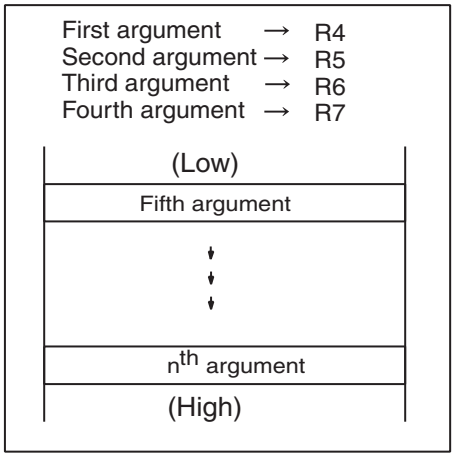


When a function receiving a variable count of arguments is to be called, the arguments are placed in registers in the same manner as for transfer. The called function stores all the register-delivered arguments in the argument register save area in the stack.

The argument area is allocated/deallocated by the caller function, whereas the argument register save area is allocated/deallocated by the callee function.

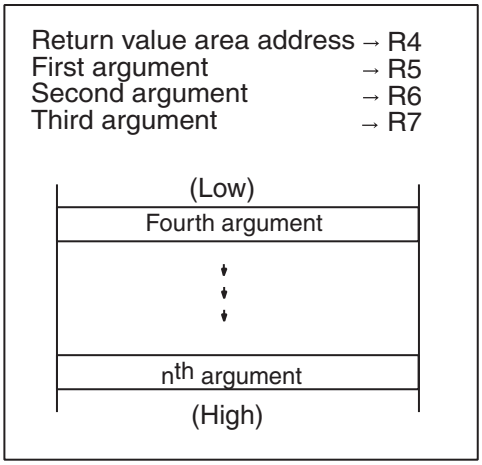
Figure 4.6-2 and Figure 4.6-3 show the argument formats prescribed in the standard linkage regulations.

Figure 4.6-2 fcc911s Command Argument Format Stated in Standard Linkage Regulations



Note: Two argument registers are required for 8-byte type arguments.

Figure 4.6-3 Argument Format for fcc911s Command Structure/Union Return Function Calling



Note: Two argument registers are required for 8-byte type arguments.

4.6.3 fcc911s Command Argument Extension Format

When an argument is to be stored in the stack, its type is converted to an extended type in accordance with the individual argument type. The argument is freed by the caller function after the return from the callee function is made.

■ fcc911s Command Argument Extension Format

Table 4.6-1 shows the argument extension format.

Table 4.6-1 fcc911s Command Argument Extension Format

Actual Argument Type	Extended Type *1	Stack Storage Size [Byte]
_Bool	int	4
char	int	4
signed char	int	4
unsigned char	int	4
short	int	4
unsigned short	int	4
int	No extension	4
unsigned int	No extension	4
long	No extension	4
unsigned long	No extension	4
float	double	8
double	No extension	8
long double	No extension	8
pointer/address	No extension	4
Structure/union	-	4 *2
Class	-	4 *3

*1: The extended type represents an extended type that is provided when no argument type is given. When a prototype declaration is made, it is complied with.

*2: When a structure/union is to be delivered as an argument, the caller copies it to the local variable area and delivers the address of that area.

*3: Passing of a class as an argument depends on the availability and contents of a copy constructor. Without the copy constructor, the class is passed in the same manner as the structure.

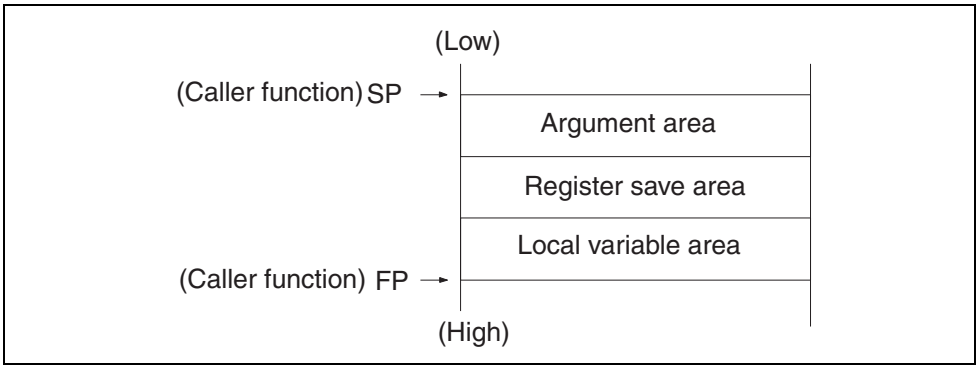
4.6.4 fcc911s Command Calling Procedure

The caller function initiates branching to the callee function after argument storage.

■ fcc911s Command Calling Procedure

Figure 4.6-4 show the stack frame prevailing at calling in compliance with the standard linkage regulations.

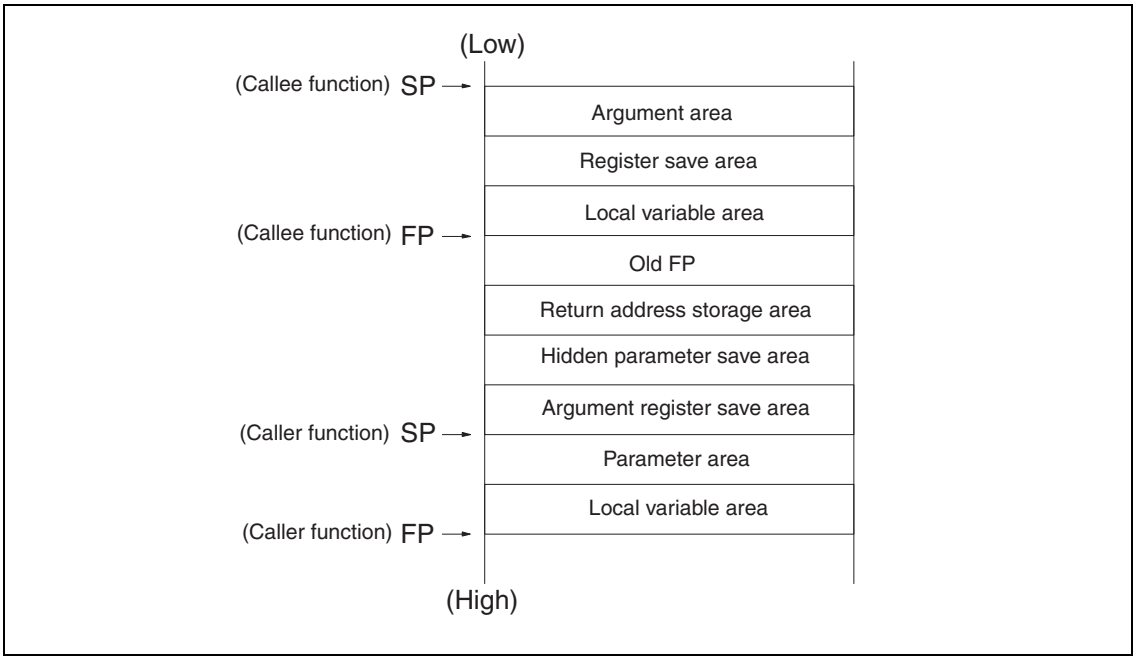
Figure 4.6-4 Stack Frame Prevailing at Calling in Compliance with fcc911s Command Standard Linkage Regulations



The callee function saves the caller function frame pointer (FP) in the stack and then stores the prevailing stack pointer value in the stack as the new frame pointer value. Subsequently, the local variable area and caller function register save area are acquired from the stack to save the caller register.

Figure 4.6-5 show the stack frame that is created by the callee function in compliance with the standard linkage regulations.

Figure 4.6-5 Stack Frame Created by Callee Function in Compliance with fcc911s Command Standard Linkage Regulations



4.6.5 fcc911s Command Register

This section states the register guarantee and register setup regulations in the standard linkage regulations.

■ fcc911s Command Register Guarantee

The callee function guarantees the following registers of the caller function.

- General-purpose registers R8 to R11, R14, and R15
- Floating point registers FR8 to FR15

The register guarantee is provided when the callee function acquires a new area from the stack and saves the register value in that area. Note, however, that registers remaining unchanged within the function are not saved. If such registers are altered using the ASM statement, etc., no subsequent operations will be guaranteed.

■ fcc911s Command Register Setup

The register regulations for function call and return periods are indicated in Table 4.6-2.

Table 4.6-2 Register Regulations for fcc911s Command Function Call and Return Periods

Register	Call Period	Return period
R4	Argument/return value area address*1	Return value*2
R5	Argument register*1	Return value*3
R6 and R7	Argument register*1	Not stipulated
R0 to R3	Not stipulated	Not stipulated
R12 and R13	Not stipulated	Not stipulated
R8 to R11	Not stipulated	Call period value guaranteed
R14	Frame pointer (FP)	Call period value guaranteed
R15	Stack pointer (SP)	Call period value guaranteed
FR0 to FR7 *4	Not stipulated	Not stipulated
FR8 to FR15 *4	Not stipulated	Call period value guaranteed

*1: There are no stipulations for unused registers in situations where the argument is less than 4 words.

*2: There are no stipulations for situations where a function without the return value is called or a function with a structure/union type return value is called.

*3: There are no stipulations for situations where the function to be called has a return value other than a double or long double type.

*4: These exist only in FR81 with FPU.

4.6.6 fcc911s Command Return Value

The return value interface stated in the standard linkage regulations is indicated in Table 4.6-3.

■ fcc911s Command Return Value

Table 4.6-3 fcc911s Command Return Value Interface Stated in Standard Linkage Regulations

Return Value Type	Return Value Interface
void	None
_Bool	R4
char	R4
signed char	R4
unsigned char	R4
short	R4
unsigned short	R4
int	R4
unsigned int	R4
long	R4
unsigned long	R4
float	R4
double	R4 and R5 *1
long double	R4 and R5 *1
Pointer/address	R4
Structure/union	R4 *2
Class	R4 *3

*1: The 4 high-order bytes of a total of 8 bytes are stored in R4 and the remaining 4 low-order bytes are stored in R5.

*2: When a structure/union is used as the return value, the caller function stores the start address of the return value storage area into R4 and then passes it to the callee function. The callee function interprets R4 as the start address of the return value storage area. When this address needs to be saved in memory, the callee function secures the hidden parameter save area and saves the address in that area.

*3: Passing of a class as a return value depends on the availability and contents of a copy constructor. Without the copy constructor, the class is passed in the same manner as the structure.

4.7 fcc911s Command Interrupt Function Call Interface

The interrupt function can be written using the __interrupt type qualifier. If the interrupt function is called by a method other than an interrupt, no subsequent operations will be guaranteed. The function call interface within the interrupt function is the same as stated in the standard linkage regulations.

■ fcc911s Command Interrupt Function Call Interface

● Interrupt stack frame

When an interrupt occurs, the stack is changed to the interrupt stack.

● Argument

No argument can be specified for the interrupt function. If any argument is specified for the interrupt function, no subsequent operations will be guaranteed.

● Interrupt function calling procedure

The interrupt function is called by an interrupt via the interrupt vector table. If the interrupt function is called by any other method, no subsequent operations will be guaranteed.

● Register

As regards the interrupt function, all registers are guaranteed. When -K NOSAVEFREG is specified on FR81 with FPU, the value from FR0 to FR15 and FCR is not guaranteed.

● Return value

Interrupt function does not usually have a return value.

4.7.1 fcc911s Command Interrupt Stack Frame

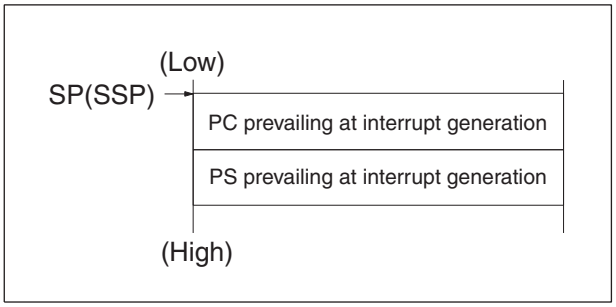
When an interrupt occurs, the stack is changed to the interrupt stack.

■ Interrupt Stack Frame

When an interrupt occurs, the stack pointer (SP) is replaced by the interrupt stack pointer (SSP). Within the interrupt function, the interrupt stack pointer is used as the normal stack pointer.

Figure 4.7-1 shows the interrupt stack frame status prevailing immediately after interrupt generation.

Figure 4.7-1 fcc911s Command Interrupt Stack Frame



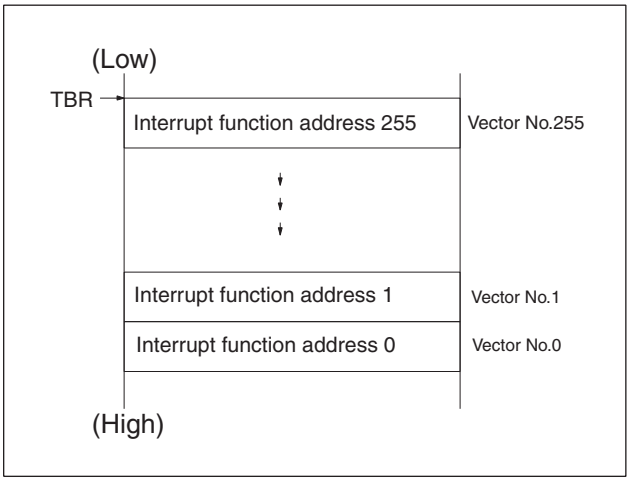
4.7.2 fcc911s Command Interrupt Function Calling Procedure

The interrupt function is called by an interrupt via the interrupt vector table. If the interrupt function is called by any other method, no subsequent operations will be guaranteed.

■ Interrupt Function Calling Procedure

Figure 4.7-2 shows an example of interrupt vector table.

Figure 4.7-2 fcc911s Command Interrupt Vector Table



When an interrupt is generated, the vector table corresponding to the interrupt vector number is referenced according to the following calculation.

TBR + 0x3FC - (4 × vector number)

CHAPTER 5

EXTENDED LANGUAGE SPECIFICATIONS

This chapter explains about the extended language specifications supported by the compiler. The limitations on compiler translation are also described in this chapter.

5.1 Assembler Description Functions

There are the following two assembler description functions.
1 asm statement
2 Pragma instruction

■ Description by asm Statement

When the asm statement is written, the character string literal is expanded as the assembler instruction. This function makes it possible to write the asm statement inside and outside the function.

[General format]

_asm (Character string literal);

[Explanation]

When the asm statement is written inside the function, the assembler is expanded at the written position.

When the statement is written outside the function, it is expanded as an independent section. Therefore, if the statement is to be written outside the function, be sure to write the section definition pseudo instruction to define the section. If the section is not defined, no subsequent operations will be guaranteed.

When using a general-purpose register within the asm statement in the function during fcc911s command execution, the user is responsible for register saving and restoration. However, the user need not do the register saving and restoretion to general-purpose registers R0 to R3, R12, and R13.

If the asm statement exists in a C/C++ source program, a part of optimization is stopped even when the -O optimization option is specified.

[Example]

```
Input:  /* When written inside the function */
        extern int temp;
        sample( ){
           _ _asm("      LDI    #1, R0");
           _ _asm("      LDI:32 #_temp, R12");
           _ _asm("      ST     R0, @R12");
          }
        /* When written outside the function */
        _ _asm("      .SECTION    DATA, DATA, ALIGN=4");
        _ _asm("      .GLOBAL _a");
        _ _asm("_a:");
        _ _asm("      .RES.B 4");

Output:          .SECTION          CODE, CODE, ALIGN=2
        ;-------begin_of_function
                 .GLOBAL _sample
        _sample:
                 ST     RP, @-SP
                 ENTER  #4
                 LDI    #1, R0
                 LDI:32 #_temp, R12
```

```
        ST      R0, @R12
L_sample:
        LEAVE
        LD      @SP+, RP
        RET
        .SECTION            DATA, DATA, ALIGN=4
        .GLOBAL _a
 _a:
        .RES.B 4
```

■ Description by Pragma Instruction

The descriptions between "#pragma asm" and "#pragma endasm" are expanded directly as the assembler instruction. This function makes it possible to write the statement inside and outside the function.

[General format]

```
#pragma  asm
      Assembler description
#pragma  endasm
```

[Explanation]

When the statement is written inside the function, the assembler is expanded at the written position.

When the statement is written outside the function, it is expanded as an independent section. Therefore, if the statement is to be written outside the function, be sure to write the section definition pseudo instruction to define the section. If the section is not defined, no subsequent operations will be guaranteed.

When using a general-purpose register within the asm statement in the function during fcc911s command execution, the user is responsible for register saving and restoration. However, the user need not do the register saving and restoretion to general-purpose registers R0 to R3, R12, and R13.

If the assembler provided by #pragma asm/endasm exists in the C source program, a part of optimization is stopped even when the -O optimization option is specified.

[Example]

```
Input:  /* When written inside the function */
        extern int temp;
        sample( ){
        #pragma   asm
                  LDI     #1, R0
                  LDI:32  #_temp, R12
                  ST      R0, @R12
        #pragma   endasm
        }
        /* When written outside the function */
        #pragma asm
                  .SECTION          DATA, DATA, ALIGN=4
                  .GLOBAL _a
        _a:
                  .RES.B 4
        #pragma   endasm
Output:           .SECTION          CODE, CODE, ALIGN=2
        ;-------begin_of_function
             .GLOBAL _sample
        _sample:
                  ST     RP, @-SP
                  ENTER  #4
                  LDI    #1, R0
                  LDI:32 #_temp, R12
                  ST     R0, @R12
        L_sample:
                  LEAVE
                  LD     @SP+, RP
                  RET
                  .SECTION          DATA, DATA, ALIGN=4
                  .GLOBAL _a
        _a:
                  .RES.B 4
```

5.2 Interrupt Control Functions

There are the following five interrupt control functions.

1 Interrupt mask setup function
2 Interrupt mask disable function
3 Interrupt level setup function
4 Interrupt function description function
5 Interrupt vector table generation function

■ Interrupt Mask Setup Function

[General format]

void __DI(void);

[Explanation]

Expands the interrupt masking code.

[Example]

```
Input: _ _DI( );
Output:
        ANDCCR #0xef
```

■ Interrupt Mask Disable Function

[General format]

void __EI(void);

[Explanation]

Expands the interrupt masking disable code.

[Example]

```
Input: _ _EI( );
Output:
        ORCCR #0x10
```

■ Interrupt Level Setup Function

[General format]

void __set_il(int level);

[Explanation]

Expands the code for changing the interrupt level to the specified level.

[Example]

```
Input: _ _set_il(2);
Output:
        STILM #2
```

■ Interrupt Function Description Function

[General format1]

__interrupt void Interrupt function (void){ ... }

[General format2]

extern __interrupt void Interrupt function (void);

[Explanation]

The interrupt function can be written by specifying the __interrupt type qualifier. Since the interrupt function is called by an interrupt, it is impossible to set up an argument or obtain a return value. If a function declared or defined by the __interrupt type qualifier is called by performing the normal function calling procedure, no subsequent operations will be guaranteed.

[Example]

```
Input: _ _interrupt void sample(void){ ... }
Output:
        _func:
                STM     (R12, R13)
                ST      MDH, @-SP
                ST      MDL, @-SP
                ST      RP, @-SP
                ENTER   #4
                ....
        L_func:
                LEAVE
                LD      @SP+, RP
                LD      @SP+, MDL
                LD      @SP+, MDH
                LDM     (R12, R13)
                RETI
```

■ Interrupt Vector Table Generation Function

[General format]

#pragma intvect [Interrupt function name |32-bit unsigned constant] Vector number

#pragma defvect Interrupt function name

[Explanation]

#pragma intvect generates an interrupt vector table for which the interrupt function is set.

For product that can set the reset mode, the reset mode can be specified by setting the mode value to vector 1. Please see the LSI specification Manual for details of the reset mode.

#pragma defvect specifies the default interrupt function to be set for interrupt vectors not specified by #pragma intvect.

The interrupt vector table is generated in an independent section named INTVECT.

All interrupt vector tables must be defined using the same translation unit (file). If #pragma intvect or #pragma defvect is specified using two or more translation units, no subsequent operations will be guaranteed.

The definition cannot be formulated two or more times for the same vector number. However, no error occurs if the definitions are identical.

No value other than an integer constant may be specified as the vector number. Specify a vector number between 0 and 255.

The reset vectors must always be assigned to 0xFFFFC. For this reason, to set TBR to other than 0xFFC00, use the asm statement to define reset vectors separately.

The mode value for the reset vector must always be assigned to 0xFFFF8. The vector number corresponding to 0xFFFF8 is vector 1.

Caution:

Please set the reset mode to the most significant byte of 32-bit constant value set to vector 1. The value set to three subordinate position bytes is disregarded.

[Example]

```
Please describe the reset mode as follows when the reset mode is 5.
Input:
     extern __interrupt void startup();
     #pragma intvect  startup 0          // The reset vector
     #pragma intvect  0x05000000 1       // The reset mode
Output:
       .SECTION     INTVECT, CONST, ALIGN=4
       .ALIGN       4
       .DATAB.W     254, 0
       .DATA.W      0x5000000     ;; The reset mode is 5
       .DATA.W      _startup      ;; The reset vector
```

5.3 I/O Area Access Function

The I/O area operation variable can be defined by specifying the __io type qualifier.

■ I/O Area Access Function

[General format]

__io Variable definition;

[Explanation]

The fcc911s command enables the definition of variables operating the I/O area defined between addresses 0x00 and 0xff by specifying the __io type qualifier. The fcc911s command makes variables available up to address 0x3ff, depending on their type. Since a highly efficient dedicated instruction is provided for I/O area access, a higher-speed, more-compact object can be generated. This instruction cannot be used for variables operating an I/O area positioned at addresses higher than 0xff. To define a variable that accesses such an area, use the volatile type qualifier.

The initial value cannot be specified for variables for which the __io type qualifier is specified.

When the specified variable is for a structure or union, it is assumed that all members are positioned in the I/O area. The variable cannot be specified for structure or union members. For the variable for which the __io type qualifier is specified, compilation is conducted on the assumption that the volatile type qualifier is specified.

When the -K NOVOLATILE option is specified, the volatile type qualifier is not assumed to be specified for the variable for which the __io type qualifier is specified.

When the __io type qualifier is specified for an automatic variable, the variable is not treated as a variable positioned in the I/O area but an automatic variable for which the volatile type qualifier is specified.

[Example]

```
Input:   #pragma section IO=IOA ,attr=IO ,locate=0x10
            _ _io int a;
         void func(void){ a=1;}
Output:
              .SECTION     IOA ,IO ,LOCATE=H'00000010
              .GLOBAL _a
         _a:
              .RES.B 4
              .SECTION     CODE, CODE, ALIGN=2
         ;-------begin_of_function
              .GLOBAL _func
         _func:
             ST     RP, @-SP
             ENTER  #4
             LDI    #1, R0
             MOV    R0, R13
             DMOV R13, @_a
         L_func:
             LEAVE
             LD     @SP+, RP
             RET
```

5.4 In-line Expansion Specifying Function

This function specifies the user definition function for in-line expansion. In-line expansion can be specified with the -x option.

■ In-line Expansion Specifying Function

[General format]

#pragma inline Function name [, Function name...]

[Explanation]

Recursively called functions cannot be subjected to in-line expansion. It should also be noted that functions may not be subjected to in-line expansion depending on asm statement use, setjmp function calling, and other conditions.

When there are two or more descriptions for the same translation unit or in-line expansion is specified by an option, all the specified function names are valid.

The in-line expansion specifying is invalid if the -O option is not specified.

5.5 Section Name Change Function

This function is used to change the section name or section attribute and sets the section arrangement address.

■ Section Name Change Function(#pragma section)

[General format]

#pragma section DEFSECT[=NEWNAME][,attr=SECTATTR][,locate=ADDR]

[Explanation]

Change the section name to be output by the compiler from DEFSECT to NEWNAME and the section type to SECTATTR.

The locate address may also be specified as ADDR.

When "=NEWNAME" is omitted, the section name is not changed. Depending on DEFSECT, some section type is invalid. Please do not put any blanks before and behind =.

For the section name to be output by the compiler, see section "4.1 Section Structure of fcc911s Command"; for the section type that can be output, refer to the Assembler Manual.

When the locate address is given, it cannot be specified for the section at linking.

Caution:

The #pragma section affects the entire source, regardless of the location. If DEFSECT is specified many times, the last one is valid. If DEFSECT is specified by the -s option, it takes priority over the others.

The EXT_CTOR_DTOR section cannot be specified and its output is fixed.

[Example]

```
Input:   #pragma section CODE=program,attr=CODE,locate=0xff
         void main(void){}
Output:
                 .SECTION   program, CODE, LOCATE=H'000000FF,
         ;-------begin_of_function
                 .GLOBAL _main
         _main:
                 ST     RP, @-SP
                 ENTER  #4
         L_main:
                 LEAVE
                 LD     @SP+, RP
                 RET
```

■ **Section Name Change Function**(#pragma segment)

[General format]

#pragma segment DEFSECT[=NEWNAME][,attr=SECTATTR][,locate=ADDR]

[Explanation]

The section name output by the compiler is changed from DEFSECT to NEWNAME and the section type is changed to SECTATTR.

When "=NEWNAME" is omitted, the section name is not changed. Depending on DEFSECT, some section type is invalid. Please do not put any blanks before and behind =.

For the section name to be output by the compiler, see section "4.1 Section Structure of fcc911s Command"; for the section type that can be output, refer to the Assembler Manual.

The #pragma segment acts on the function definition, the variable definition and the variable declaration since the described line. This specification is effective until the #pragma segment of same next DEFSECT is described. (The description of the #pragma segment that DEFSECT is different does not influence mutually.)

When #pragma segment without NEWNAME is described, the section name of DEFSECT since the line becomes the section name of default.

When neither the function definition, the variable definition nor the variable declaration on which it acts since the line where the #pragma segment is described are defined, the #pragma segment is disregarded.

The #pragma section and -s option of the section alone not specified by the #pragma segment act when the #pragma segment, the #pragma section or -s option is specified at the same time.

The INTVECT section cannot specify.

[Example]

```
Input:
     #pragma segment CODE=program1
     void func1(void){}
     #pragma segment DATA=ram1
     int a1;
     #pragma segment CODE=program2
     void func2(void){}
     #pragma segment DATA=ram2
     int a2;
Output:
          .SECTION ram2, DATA, ALIGN=4
          .GLOBAL _a2
     _a2:
          .RES.B 4
          .SECTION ram1, DATA, ALIGN=4
          .GLOBAL _a1
     _a1:
          .RES.B 4
          .SECTION program1, CODE, ALIGN=2
          .GLOBAL _func1
     _func1:
          RET
          .SECTION program2, CODE, ALIGN=2
          .GLOBAL _func2
     _func2:
          RET
```

Caution:

#pragma segment works on the position of the first variable definition/variable declaration in the file.

Please direct the variable declaration the change in the section name if there is a variable declaration before the variable definition.

[Example]

```
Input:
        #pragma segment CONST=const1,attr=CONST,locate=0xff00
        extern const int var;   //Variable declaration
        #pragma segment CONST=const2,attr=CONST,locate=0xff10
        const int var=10;       //Variable definition

        #pragma segment CODE=program1,attr=CODE,locate=0xff20
        extern void func(void); //Function declaration
        #pragma segment CODE=program2,attr=CODE,locate=0xff30
        void func(void){}       //Function definition
Output section of variable/function
```

Variable/function name	Output section name
_var	const1
_func	program2

5.6 Interrupt Level Setup Function

This function is used to set the function interrupt level.

■ Interrupt Level Setup Function

[General format]

#pragma ilm(NUM)

#pragma noilm

[Explanation]

#pragma ilm specifies the interrupt level for the subsequently defined function.

#pragma noilm clears the interrupt level specifying.

When #progma ilm is described in the function, the interrupt level of the function is set. When #progma noilm is described in the function, the interrupt level of the function is not set.

At fcc911s command, an integer constant between 0 and 31 can be specified in the NUM position. A hexadecimal, octal, or decimal number can be described.

Although the interrupt level is changed at the beginning of the specified function, remember that the new interrupt level does not revert to the previous level at completion of function execution.

Always specify #pragma ilm and #pragma noilm as a set. Nesting is not possible.

[Example]

```
Input:  #pragma ilm(1)
        void func(void){}
        #pragma noilm
Output:
       _func:
             STILM  #1
             ST     RP, @-SP
             ENTER  #4
       L_func:
             LEAVE
             LD     @SP+, RP
             RET
```

5.7 Intrinsic Function

The following intrinsic functions are available.

- **__wait_nop**
- **Integer operation intrinsic function**

■ __wait_nop Intrinsic Function

[General format]

void __wait_nop(void);

[Explanation]

To properly time I/O access and interrupt generation, formerly, the NOP instruction was inserted using the asm statement. However, when such a method is used, the asm statement may occasionally inhibit various forms of optimization and greatly degrade the file object efficiency.

When the __wait_nop() intrinsic function is written, the compiler outputs one NOP instruction to the function call entry position. If the function call entry is performed a count of times until all the issued NOP instructions are covered, timing control is exercised to minimize the effect on optimization.

[Example]

```
Input: void sample(void){__wait_nop();}
Output:
        _sample:
                ST      RP, @-SP
                ENTER   #4
                NOP
        L_sample:
                LEAVE
                LD      @SP+, RP
                RET
```

5.7.1 Integer Operation Intrinsic Function

This function is used to the integer operation instructions supported by the FR family CPU.
To use the integer operation intrinsic function, always include the header file (builtin.h) for integer operation intrinsic function. When the header file is not included, the function cannot be recognized as the intrinsic function.

- **__mulsh (Signed 16-bit Multiply)**
- **__muluh (Unsigned 16-bit Multiply)**
- **__muls (Signed 32-bit Multiply)**
- **__mulu (Unsigned 32-bit Multiply)**
- **__divsb (Signed 8-bit Division)**
- **__divub (Unsigned 8-bit Division)**
- **__divsh (Signed 16-bit Division)**
- **__divuh (Unsigned 16-bit Division)**
- **__modsb (Signed 8-bit Modulo)**
- **__modub (Unsigned 8-bit Modulo)**
- **__modsh (Signed 16-bit Modulo)**
- **__moduh (Unsigned 16-bit Modulo)**

■ __mulsh Intrinsic Function

[General Format]

long __mulsh(signed short a, signed short b);

[Explanation]

This intrinsic function multiplies signed 16-bit data by signed 16-bit data to return a signed 32-bit result. It is possible to multiply it by CPU MULH instruction by the use of this intrinsic function.

[Example]

```
Input:  #include <builtin.h>
        extern signed short arg1, arg2;
        extern long ans;
        void sample(void) {
           ans = __mulsh(arg1, arg2);
        }
Output: LDI:32  #_arg1,R4
        LDI:32  #_arg2,R3
        LDUH    @R4,R2  ; _arg1
        LDUH    @R3,R1  ; _arg2
```

```
        EXTSH   R2
        EXTSH   R1
        MULH    R1,R2
        LDI:32  #_ans,R2
        MOV     MDL,R12
        ST      R12,@R2
```

■ __muluh Intrinsic Function

[General Format]

unsigned long __muluh(unsigned short a, unsigned short b);

[Explanation]

This intrinsic function multiplies unsigned 16-bit data by unsigned 16-bit data to return an unsigned 32-bit result.

It is possible to multiply it by CPU MULUH instruction by the use of this intrinsic function.

[Example]

```
Input:  #include <builtin.h>
        extern unsigned short arg1, arg2;
        extern unsigned long ans;
        void sample(void) {
           ans = __muluh(arg1, arg2);
        }
Output: LDI:32  #_arg1,R2
        LDI:32  #_arg2,R1
        LDUH    @R2,R5  ; _arg1
        LDUH    @R1,R4  ; _arg2
        LDI:32  #_ans,R0
        MULUH   R4,R5
        MOV     MDL,R12
        ST      R12,@R0
```

■ __muls Intrinsic Function

[General Format]

signed long long __muls(signed long a, signed long b);

[Explanation]

This intrinsic function multiplies signed 32-bit data by signed 32-bit data to return a signed 64-bit result.

It is possible to multiply it by CPU MUL instruction by the use of this intrinsic function.

This intrinsic function is expanded by specifying the -K LONGLONG option.

[Example]

```
Input:  #include <builtin.h>
extern signed long arg1, arg2;
extern signed long long ans;
void sample(void) {
   ans = __muls(arg1, arg2);
}
Output: LDI:32  #_arg1,R12
        LD      @R12,R0      ; _arg1
        LDI:32  #_arg2,R12
        LD      @R12,R12     ; _arg2
        MUL     R12,R0
        MOV     MDL,R0
        MOV     MDH,R12
        LDI:32  #_ans,R1
        ST      R12,@R1      ; _ans
        LDI     #4,R13
        ST      R0,@(R13,R1) ; _ans
```

■ __mulu Intrinsic Function

[General Format]

unsigned long long __mulu(unsigned long a, unsigned long b);

[Explanation]

This intrinsic function multiplies unsigned 32-bit data by unsigned 32-bit data to return a unsigned 64-bit result.

It is possible to multiply it by CPU MUL instruction by the use of this intrinsic function.

This intrinsic function is expanded by specifying the -K LONGLONG option.

[Example]

```
Input:  #include <builtin.h>
        extern unsigned long arg1, arg2;
        extern unsigned long long ans;
        void sample(void) {
           ans = __mulu(arg1, arg2);
        }
Output: LDI:32   #_arg1,R12
        LD       @R12,R0      ; arg1
        LDI:32   #_arg2,R12
        LD       @R12,R12    ; _arg2
        MUL      R12,R0
        MOV    MDL,R0
        MOV    MDH,R12
        LDI:32 #_ans,R1
        ST     R12,@R1      ; _ans
        LDI    #4,R13
        ST     R0,@(R13,R1) ; _ans
```

■ __divsb Intrinsic Function

[General Format]

signed short __divsb(signed char a, signed char b);

[Explanation]

This intrinsic function performs a division between signed 8-bit data and signed 8-bit data to return a signed 16-bit result.

It is possible to divide by the step DIV instruction of CPU by the using this intrinsic function.

[Example]

```
Input:  #include <builtin.h>
        extern signed char arg1, arg2;
        extern short ans;
        void sample(void) {
           ans = __divsb(arg1, arg2);
        }
Output: LDI:32  #_arg1,R4
        LDUB    @R4,R1  ; _arg1
        LDI:32  #_arg2,R3
        EXTSB   R1
        LDUB    @R3,R2  ; _arg2
        LSL     #24,R1
        MOV     R1,MDL
        EXTSB   R2
        DIV0S   R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV2    R2
        DIV3
        LDI:32  #_ans,R3
        DIV4S
        MOV     MDL,R0
        STH     R0,@R3
```

■ __divub Intrinsic Function

[General Format]

unsigned char __divub(unsigned char a, unsigned char b);

[Explanation]

This intrinsic function performs a division between unsigned 8-bit data and unsigned 8-bit data to return an unsigned 8-bit result.

It is possible to divide by the step DIV instruction of CPU by the using this intrinsic function.

[Example]

```
Input:  #include <builtin.h>
        extern unsigned char arg1, arg2;
        extern unsigned char ans;
        void sample(void) {
           ans = __divub(arg1, arg2);
        }
Output: LDI:32  #_arg1,R3
        LDUB    @R3,R1  ; _arg1
        LDI:32  #_arg2,R2
        LSL     #24,R1
        LDUB    @R2,R5  ; _arg2
        MOV     R1,MDL
        DIV0U   R5
        DIV1    R5
        DIV1    R5
        DIV1    R5
        DIV1    R5
        DIV1    R5
        DIV1    R5
        DIV1    R5
        DIV1    R5
        DIV1    R5
        LDI:32  #_ans,R2
        MOV     MDL,R0
        STB     R0,@R2
```

■ __divsh Intrinsic Function

[General Format]

signed long __divsh(signed short a, signed short b);

[Explanation]

This intrinsic function performs a division between signed 16-bit data and signed 16-bit data to return a signed 32-bit result.

It is possible to divide by the step DIV instruction of CPU by the using this intrinsic function.

[Example]

```
Input:  #include <builtin.h>
        extern signed short arg1, arg2;
        extern long ans;
        void sample(void) {
           ans = __divsh(arg1, arg2);
        }
Output: LDI:32  #_arg1,R4
        LDUH    @R4,R1  ; _arg1
        LDI:32  #_arg2,R3
        EXTSH   R1
        LDUH    @R3,R2  ; _arg2
        LSL     #16,R1
        MOV     R1,MDL
        EXTSH   R2
        DIV0S   R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV2    R2
        DIV3
        LDI:32  #_ans,R2
        DIV4S
        MOV     MDL,R12
```

```
                ST      R12,@R2
```

■ __divuh Intrinsic Function

[General Format]

unsigned short __divuh(unsigned short a, unsigned short b);

[Explanation]

This intrinsic function performs a division between unsigned 16-bit data and unsigned 16-bit data to return an unsigned 16-bit result.

It is possible to divide by the step DIV instruction of CPU by the using this intrinsic function.

[Example]

```
Input:  #include <builtin.h>
        extern unsigned short arg1, arg2;
        extern unsigned short ans;
        void sample(void) {
           ans = __divuh(arg1, arg2);
        }
Output: LDI:32 #_arg1,R3
        LDUH   @R3,R1  ; _arg1
        LDI:32 #_arg2,R2
        LSL    #16,R1
        LDUH   @R2,R5  ; _arg2
        MOV    R1,MDL
        DIV0U  R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        LDI:32 #_ans,R2
        MOV    MDL,R0
        STH    R0,@R2
```

■ __modsb Intrinsic Function

[General Format]

signed char __modsb(signed char a, signed char b);

[Explanation]

This intrinsic function performs a modulo operation between signed 8-bit data and signed 8-bit data to return a signed 8-bit result.

It is possible to divide by the step DIV instruction of CPU by the using this intrinsic function.

[Example]

```
Input:  #include <builtin.h>
        extern signed char arg1, arg2;
        extern signed char ans;
        void sample(void) {
           ans = __modsb(arg1, arg2);
        }
Output: LDI:32  #_arg1,R4
        LDUB    @R4,R1  ; _arg1
        LDI:32  #_arg2,R3
        EXTSB   R1
        LDUB    @R3,R2  ; _arg2
        LSL     #24,R1
        MOV     R1,MDL
        EXTSB   R2
        DIV0S   R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV2    R2
        DIV3
        LDI:32  #_ans,R3
        DIV4S
        MOV     MDH,R0
        STB     R0,@R3
```

■ __modub Intrinsic Function

[General Format]

unsigned char __modub(unsigned char a, unsigned char b);

[Explanation]

This intrinsic function performs a modulo operation between unsigned 8-bit data and unsigned 8-bit data to return an unsigned 8-bit result.

It is possible to divide by the step DIV instruction of CPU by the using this intrinsic function.

[Example]

```
Input:  #include <builtin.h>
        extern unsigned char arg1, arg2;
        extern unsigned char ans;
        void sample(void) {
           ans = __modub(arg1, arg2);
        }
Output: LDI:32 #_arg1,R3
        LDUB   @R3,R1  ; _arg1
        LDI:32 #_arg2,R2
        LSL    #24,R1
        LDUB   @R2,R5  ; _arg2
        MOV    R1,MDL
        DIV0U  R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        DIV1   R5
        LDI:32 #_ans,R2
        MOV    MDH,R0
        STB    R0,@R2
```

■ __modsh Intrinsic Function

[General Format]

signed short __modsh(signed short a, signed short b);

[Explanation]

This intrinsic function performs a modulo operation between signed 16-bit data and signed 16-bit data to return a signed 16-bit result.

It is possible to divide by the step DIV instruction of CPU by the using this intrinsic function.

[Example]

```
Input:  #include <builtin.h>
        extern signed short arg1, arg2;
        extern signed short ans;
        void sample(void) {
           ans = __modsh(arg1, arg2);
        }
Output: LDI:32  #_arg1,R4
        LDUH    @R4,R1  ; _arg1
        LDI:32  #_arg2,R3
        EXTSH   R1
        LDUH    @R3,R2  ; _arg2
        LSL     #16,R1
        MOV     R1,MDL
        EXTSH   R2
        DIV0S   R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV1    R2
        DIV2    R2
        DIV3
        LDI:32  #_ans,R3
        DIV4S
        MOV     MDH,R0
```

```
                STH     R0,@R3
```

■ __moduh Intrinsic Function

[General Format]

unsigned short __moduh(unsigned short a, unsigned short b);

[Explanation]

This intrinsic function performs a modulo operation between unsigned 16-bit data and unsigned 16-bit data to return an unsigned 16-bit result.

It is possible to divide by the step DIV instruction of CPU by the using this intrinsic function.

[Example]

```
    Input:  #include <builtin.h>
            extern unsigned short arg1, arg2;
            extern unsigned short ans;
            void sample(void) {
               ans = __moduh(arg1, arg2);
            }
    Output: LDI:32  #_arg1,R3
            LDUH    @R3,R1  ; _arg1
            LDI:32  #_arg2,R2
            LSL     #16,R1
            LDUH    @R2,R5  ; _arg2
            MOV     R1,MDL
            DIV0U   R5
            DIV1    R5
            DIV1    R5
            DIV1    R5
            DIV1    R5
            DIV1    R5
            DIV1    R5
            DIV1    R5
            DIV1    R5
            DIV1    R5
            DIV1    R5
            DIV1    R5
            DIV1    R5
            DIV1    R5
            DIV1    R5
            DIV1    R5
            DIV1    R5
            DIV1    R5
            DIV1    R5
            LDI:32  #_ans,R2
            MOV     MDH,R0
            STH     R0,@R2
```

5.8 Predefined Macros

Macro names predefined by the compiler are explained below.

■ Macros Stipulated by ANSI Standard

The ANSI standard stipulates the macros listed in Table 5.8-1.

Table 5.8-1 Macros stipulated by the ANSI standard

Macro	Description
__LINE__	Defines line number of current source line.
__FILE__	Defines source file name.
__DATE__	Defines source file translation date.
__TIME__	Defines source file translation time.
__STDC__	Macro indicating that the processing system meets requirements. When the -Ja option is specified, 0 is selected as the definition. When the -Jc option is specified, 1 is selected as the definition.

In addition to the macros listed in Table 5.8-1, C++ has the macros listed in Table 5.8-2.

Table 5.8-2 C++ macros (in addition to the macros stipulated by the ANSI standard)

Macro	Description
__cplusplus	1 is defined.
c_plusplus	1 is defined. Nothing is defined when the -Jc option is specified.
__embedded_cplusplus	1 is defined only when the default -Je option is specified.

■ Macros Predefined by fcc911s Command

The fcc911s command predefines the following macros.

Table 5.8-3 Macros predefined by fcc911s command

Macro	Description
__COMPILER_FCC911__	1 is defined.
__CPU_ MB number__	1 is defined. "MB number" of macro name is actually the MB number specified by the -cpu option.
__CPU_FR__ __CPU_FR80__ __CPU_FR81__ __CPU_FR81_FPU__	Either of them is defined as 1, depending on the MB number specified by the -cpu option.

5.9 Limitations on Compiler Translation

Table 5.9-1 shows the translation limitations to be imposed when the compiler is used. The table also indicates the minimum ANSI standard to be met.

■ Limitations on Compiler Translation

Table 5.9-1 List of Translation Limitations (1 / 2)

No.	Function	ANSI Standard	Compiler
1	Count of nesting levels for a compound statement, repetition control structure, and selection control structure	15	infinity
2	Count of nesting levels for condition incorporation	8	infinity
3	Count of pointers, arrays, and function declarators (any combinations of these) for qualifying one arithmetic type, structure type, union type, or incomplete type in a declaration	12	infinity
4	Count of nests provided by parentheses for one complete declarator	31	infinity
5	Count of nest expressions provided by parentheses for one complete expression	32	infinity
6	Count of valid leading characters of internal identifier or macro name	31	1024
7	Count of valid leading characters of external identifier	6	1024
8	Count of external identifiers of one translation unit	511	infinity
9	Count of identifiers having the block valid range in one block	127	infinity
10	Count of macro names that can be simultaneously defined by one translation unit	1024	infinity
11	Count of virtual arguments in one function definition	31	infinity
12	Count of actual arguments for one function call	31	infinity
13	Count of virtual arguments in one macro definition	31	infinity
14	Count of actual arguments in one macro call	31	infinity
15	Maximum count of characters in one logical source line	509	infinity
16	Count of characters in a (linked) byte character string literal or wide-angle character string literal (terminal character included)	509	infinity
17	Count of bytes of one arithmetic unit	32767	infinity
18	Count of nesting levels for #include file	8	infinity

Table 5.9-1 List of Translation Limitations (2 / 2)

No.	Function	ANSI Standard	Compiler
19	Count of case name cards in one switch statement (excluding nested switch statements)	257	infinity
20	Count of members of one structure or union	127	infinity
21	Count of enumerated type constants in one enumerated type	127	infinity
22	Count of structure or union nesting levels for one structure declaration array	15	infinity

The "infinity" in the above table indicates the dependence on the memory size available for the system.

5.10 Re-include Prevention Function

The file can have #pragma once to prevent the header file from being re-included.

■ Re-include Prevention Function

The file can have #pragma once to prevent the header file from being re-included. The include file specified by #include directive described after #progma once becomes the target of the re-include prevention.

[Example]

```
file1.h:
        #pragma once
        #include "file2.h"
file2.h:
        #pragma once
        #include "file1.h"
file3.c:
        #include "file1.h" /* file1.h file2. h are included one each. */
```

5.11 Function for Controlling Instantiation of C++ Template

#pragma instantiate forces instantiation.
#pragma do_not_instantiate does not provide instantiation.

■ Function for Controlling Instantiation of C++ Template

Instantiation of a C++ template can be controlled depending on the following settings:

- #pragma instantiate template name
- #pragma do_not_instantiate template name

The types that can be specified as template names are given below:

- Template class name: A<int>
- Template class declaration: class A<int>
- Member function name: A<int> : : f
- Static data member name: A<int> : : i
- Static data member declaration: int A<int> : : i
- Member function declaration: void A<int> : : f (int, char)
- Template function declaration: char *f(int, float)

● #pragma instantiate

The specified template is forcibly instantiated in the module.

Instantiation requires a complete template definition.

● #pragma do_not_instantiate

The specified template is not instantiated in the module.

Caution:

When executing modules, #pragma instantiate is specified only for one template in one module. In other modules, #pragma do_not_instantiate must be specified.

5.12 _Bool type

_Bool type is an unsigned integer type whose value is either 0 or 1.

■ _Bool type

[General Format]

_Bool Variable name;

_Bool Function name();

[Explanation]

_Bool type has the following language specifications:

- _Bool is a keyword which means an unsigned integer type.
- Value of _Bool type is either 0 or 1.
- Type specifiers signed/unsigned can not be specified to _Bool type.
- By integral promotion, _Bool type becomes int type and its value does not change.
- When type converting _Bool type into the other integer type explicitly, the value does not change.
- When type converting a scalar value into _Bool type, its value becomes 0 if the original value is 0 and becomes 1 otherwise.
- _Bool type can be used as a bitfield. Type of a bitfield of _Bool type is always unsigned regardless of -K {SBIT|UBIT} option. Maximum bit width is 8.
- When the sizeof operator is applied to a _Bool type variable, the operator returns 1.

By including the standard header <stdbool.h>, the following 4 macros are available:

- bool
  Expanded into _Bool
- true.
  Expanded into an integer constant 1
- false
  Expanded into an integer constant 0
- __bool_true_false_are_defined.
  Expanded into an integer constant 1

Caution:

_Bool type can be used in C sources when -Jc option is not specified.
_Bool cannot be used for identifier because it is a keyword.

[Example]

Input:

```
  #include <stdbool.h>
  int i, j;

  extern void sub1();
  extern void sub2();

/* Definition of function which returns _Bool type */
_Bool set_value(int i, int j) {
  if (i > j) {
       return true;       /* Returns the true */
  } else {
       return false;      /* Returns the false */
  }
}

void func() {
    _Bool b;              /* Declaration of a _Bool type variable */
    b = set_value(i, j);

    if (b == true) {      /* Compared to the true */
       sub1();
    } else {
       sub2();
    }
}
```

CHAPTER 6

EXECUTION ENVIRONMENT

User programs are executed with or without the existence of an operating system.
In an environment in which the operating system exists, it is necessary to prepare the setup process suitable for the environment.
This chapter describes the user program execution procedure to be performed in an environment where no operating system exists.

6.1 Execution Process Overview

In an environment where no operating system exists, it is necessary to prepare the startup routine which initiates user program execution.

■ Execution Process Overview

The main functions to be incorporated into the startup routine are as follows.

● Environment initialization necessary for program operation

This initialization must be described by the assembler and completed before user program execution.

● User program calling

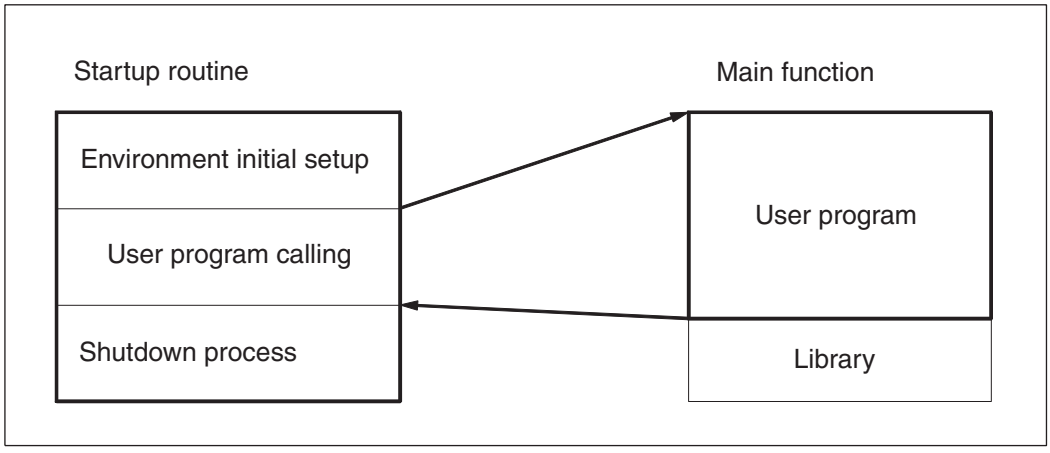
The main function, which is normally used as the function that the startup routine calls in the program start process, is to be called.

● Shutdown process

After a return from the user program is made, the shutdown process necessary for the system is to be performed to accomplish program termination.

The relationship between the startup routine and user function calling is shown in Figure 6.1-1.

Figure 6.1-1 Relationship between Startup Routine and User Function Calling



The precautions to be observed in startup routine preparation are described below.

● Stack

When the user program is executed, the stack is used for return address, argument storage area, automatic variable area, and register saving, etc. The stack must therefore be provided with an adequate space.

● Register

When the startup routine calls the user program, it is essential that stack pointer setup be completed. The user program operates on the presumption that the stack top is set as the stack pointer. Further, when the startup routine returns from the user program, the register status is as shown in Table 6.1-1. This is because the employed interface is the same as for register guarantee at the time of function calling.

For register guarantee, see section "4.6.5 fcc911s Command Register". If the guarantee of a register is called for by the system while the value of that register is not guaranteed by the user program, it is necessary to guarantee the value by the startup routine to initiate calling.

Table 6.1-1 fcc911s Command Register Status Prevailing at Return from User Program

Register	Value Guarantee at Return
R0 to R7	Not provided
R12, R13	Not provided
R8 to R11	Provided
R14 (FP)	Provided
R15 (SP)	Provided
FR0 to FR7	Not provided
FR8 to FR15	Provided

6.2 Startup Routine Creation

The processes necessary for startup routine creation are described below.

■ Startup Routine Creation

1. Register initial setup:

   Set the stack pointer (SP) to the top of the stack (stack top).

2. Data area initialization:

   The C and C++ language specification guarantees the initialization of external variables without the initial value and static variables to 0. Therefore, initialize the DATA section to 0.

3. Initialization data area duplication:

   When incorporating constant data or program into ROM, the data positioned in the ROM area needs to be copied to the RAM area. However, this duplication step is unnecessary if such a data rewrite operation will not performed within the user program. The area to be incorporated into ROM is usually positioned in the INIT section. When incorporation into ROM is specified, the linker automatically generates the following symbols for the specified section name.

   ROM_specified section name

   RAM_specified section name

   The above symbols indicate the ROM and RAM area start addresses, respectively. An example specifying of incorporation into ROM for the INIT section is shown below.

```
> fcc911s -ro ROM=ROM address range -ra RAM=RAM address range
                                    -sc @INIT=ROM ,INIT=RAM .....
```

   For the details of incorporation into ROM, refer to the Linkage Kit Manual

4. Library initial setup:

   When using the libraries, open a file for standard input/output. For details, see section "8.2 Initialization/ Termination Process Necessary for Using Library", Initialization/Termination Process Necessary for Library Use.

5. Initialization unique to C++:

   In the C++ specifications, when external or static objects are used, a constructor must be called followed by the main function. Because four-byte pointers to the main function are stored in the EXT_CTOR_DTOR section, call a constructor sequentially from the lower address of the four addresses in that section.

   In a program requiring normal termination, use the atexit function to register the address of __call_dtors function as the function to be called from the exit function. Then, call the exit function after the end of the main function.

6. User program calling:

   Call the user program.

7. Program shutdown process:

   The close process must be performed for opened files. The normal end and abnormal end processes must be prepared in accordance with the system.

■ Base Pointer Register (BP) Setting

The following settings are necessary in the startup routine when the -K BP option is specified. For the detail of this option, please see section "3.5.5 Optimization Related Options".

1. The BP relative access code uses the offset from the __bpstart.
   Define the __bpstart as follows:

```
        .export __bpstart
        .section IO__DUMMY, IO, locate=0x400
__bpstart:
        .section DATA, data, align=4
        .section INIT, data, align=4
```

2. Add the following code before calling main function.

```
        ldi:32 #__bpstart, r13
        mov r13, bp
```

Caution:

At link time, the startup routine must be linked at the head.

- When Workbench is used to make a load module, follow the instructions as follows:
  1) Choose the SRC tab in the project window and right-click the target name.
  2) Choose the "Set Linkage order" from the shortcut menu, so that the dialog is opened.
  3) Move the filename of startup routine at the top.
- When fcc911s is used from the command line to make a load module, please specify the object file name of startup routine to the argument of the -startup option.
- When flnk911s is used from the command line to make a load module, please write the name of startup routine at the head of inputs which are object module files and load module files.

Please refer to the manual of each tool for details.

■ FPU Control Register (FCR) Initialization

It is necessary to initialize FCR in the startup routine, so as to set Floating-point Exception Enabled Flags (EEF) and Rounding Mode (RM) when target CPU is FR81 with FPU.

Please add the following code before calling main function.

```
        ldi #0, r13
        mov r13, fcr
```

FCR is initialized to 0 in example above. In this case, no floating-point exceptions occur and round-to-nearest mode is specified.

Please refer to the hardware manual for details.

CHAPTER 7
LIBRARY OVERVIEW

This chapter outlines the C libraries by describing the organization of files furnished by the libraries and the relationship to the system into which the libraries are incorporated.

7.1 File Organization

This section describes the files furnished by the libraries.

■ File Organization

The following library files are provided:

● fcc911s command library files

The four fcc911s command library files below are provided.

- lib911.lib: Standard C library
- lib911if.lib: Simulator/debugger low-level function library
- lib911e.lib: EC++ library
- lib911p.lib: C++ library (having the same contents as lib911e.lib)

■ fcc911s Command Library Section Names

Table 7.1-1 shows the section names used by the fcc911s command libraries.

Table 7.1-1 fcc911s Command Library Section Names

Section Type	Section name
Code section	CODE
Data section	DATA
Initialized section	INIT
Constant section	CONST

7.2 Relationship to Library Incorporating System

This section describes the relationship between the libraries and library incorporating system.

■ System-dependent Processes

File input/output, memory management, and program termination procedures are dependent on the system. When such system-dependent processes are needed, the libraries call low-level functions (For the details of low-level functions, see "CHAPTER 8 LIBRARY INCORPORATION").

When using the libraries, prepare such low-level functions in accordance with the system.

■ Low-level Function (System-dependent Process) Types

The low-level function types and their roles are summarized below. For the detailed feature descriptions of low-level functions, see section "8.5 Low-level Function Specifications".

- open: Function for opening a file in the system
- close: Function for closing a file in the system
- read: Function for reading characters from a file
- write: Function for writing characters into a file
- lseek: Function for changing the file position
- isatty: Function for checking whether a file is a terminal file
- sbrk: Function for dynamically acquiring/changing the memory
- _exit: Function for normal program ending
- _abort: Function for abnormal program ending

■ Time Function (System-dependent Process) Types

The time function types and their roles are summarized below. For the detailed descriptions of time functions, see section "8.6 Time Function Specifications".

- clock : Function for getting the processor time consumed
- time : Function for getting the current calendar time

CHAPTER 8

LIBRARY INCORPORATION

This chapter describes the processes and functions for preparing for useing library.

8.1 Library Incorporation Overview

This section outlines library incorporation.

■ Processes and Functions must be prepared for Using Library

File input/output, memory management, and program termination procedures are the processes dependent on the system. Therefore, such processes are separated from the standard library, and whenever such processes are needed, they will be called as a low-level function. Further, the stream area initialization and other processes are necessary for using library.

The following processes and functions must be prepared for using library.

- Initialization of stream area
- The open and close processes of the standard input/output and standard error output file
- Definition of Low-level functions
- Definition of time functions

At the time of library incorporation, the above processes and functions must be prepared in accordance with the system.

8.2 Initialization/Termination Process Necessary for Using Library

This section describes the initialization/termination process required for Using Library.

■ Initialization/Termination Process

Some standard library functions require the following processes.

- Initialization of stream area
- The open and close processes of the standard input/output and standard error output file

Detailed in this section (For required functions, see section "8.4 Standard Library Functions and Required Processes/Low-level Functions".

■ Initialization of Stream Area

The _stream_init function initializes the stream area. This function must be called by the startup routine to initialize the stream area.

void _stream_init(void);

■ The Open and Close Processes of the Standard Input/Output and Standard Error Output File

Because the standard input/output and standard error output do not open or close files during the execution of programs, files must be opened before the main function is called and must be closed when the main function is completed.

Use the startup routine to perform the opening process before main function calling and the closing process after main function execution.

However, the _stream_init function correlates the file numbers 0, 1, and 2 to the stdin, stdout, and stderr streams. Therefore, the opening process need not be performed when the system's standard input, standard output, and standard error output are opened as the file numbers 0, 1, and 2.

If the system's standard input/output and standard error output are not opened or the file numbers do not match, perform the following process to open the system's files.

- freopen("Standard input name", "r", stdin);
- freopen("Standard output name", "w", stdout);
- freopen("Standard error output name", "w", stderr);

Error detection concerning the above process should be conducted as needed.

Further, the file names specified by the open function must be written as the standard input/output and standard error output names.

For the closing process, use the fclose function.

■ Time zone setting

Please set the time zone to global variable _TZ.

Please set 9*3600 for JST (Japan Standard Time) because the unit of the value set to _TZ is "second".

To set _TZ, please include time.h.

Please initialize _TZ when you use the mktime function, the ctime function and the localtime function.

In the mktime function, the ctime function and the localtime function, it is not considered that the value of _TZ is modified while executing these functions. When _TZ is modified, the result is not guaranteed.

8.3 Low-level Function Types

This section outlines the standard library functions and necessary low-level functions. The standard library functions require the following six types of low-level functions:

- **File opening and closing (open, close)**
- **Input and output relative to file (read, write)**
- **File position change (lseek)**
- **File inspection (isatty)**
- **Memory area dynamic acquisition (sbrk)**
- **Program abnormal end and normal end (_abort, _exit)**

The above processes are called from the associated standard libraries to manipulate the system's actual files or exercise program execution control.

■ Low-level Function Types

● File Opening and Closing

All functions that open the fopen function and other files call the open function to open an actual file on the system. Similarly, all functions that close the fclose function and other files call the close function to close an actual file on the system.

● Input and Output Relative to File

The scanf, printf, and other input/output functions perform input/output operations relative to the system's actual files when the read and write functions are called.

● File Position Change

The fseek and other file position manipulation functions acquire or change the system's actual file positions when the lseek function is called.

● File Inspection

The opened file is to be checked to see whether it is a terminal file.

● Memory Area Dynamic Acquisition

The malloc and other memory area dynamic acquisition functions acquire or free specific memory areas when the sbrk function is called.

● Program Abnormal End and Normal End

The abort function and exit function call the _abort function and _exit function, respectively, as the termination process.

8.4 Standard Library Functions and Required Processes/Low-level Functions

This section describes the standard library functions and associated initialization/ termination processes and low-level functions.

■ Standard Library Functions and Required Processes/Low-level Functions

Table 8.4-1 lists the standard libraries that use low-level functions, related initialization and termination processes, and low-level functions.

Table 8.4-1 Standard Library Functions and Required Processes/Low-level Functions

Standard Library Function	Low-level Function		Initialization/Termination Process
assert () abort () *	open () read () lseek () sbrk ()	close () write () isatty () _abort ()	Stream area initialization process standard input/ output and standard error output opening and closing
All stdio.h file operation functions	open () read () lseek () sbrk ()	close () write () isatty ()	Stream area initialization process standard input/ output and standard error output opening and closing
calloc () malloc () realloc () free ()	sbrk ()		
exit () *	open () read () seek () sbrk ()	close () write () isatty () _exit ()	Stream area initialization process standard input/ output and standard error output opening and closing

* : Then the abort function and exit function are called, they perform the closing process for open files. Therefore, the file manipulation related low-level functions (open, close, read, write, lseek, and sbrk) and stream area initialization and like processes are required.

In a program that is not using a file, the _abort function can be directly called instead of the abort function.

In a program for which function registration is not completed using the atexit function, the _exit function can be directly called instead of the exit function while no file is being used.

In the above instances, file manipulation related low-level function use and stream area initialization are not required.

8.5 Low-level Function Specifications

There are various low-level functions. The open, close, read, write, lseek, and isatty functions provide file processing. The sbrk function provides memory area dynamic allocation. The _exit or _abort function is used to terminate a program by calling the exit or abort function. These low-level functions must be created to suit the system.

■ Low-level Function Specifications

Create the low-level functions in compliance with the specifications stated in this section.

8.5.1 open Function

The open function should be generated according to the specifications in this section.
#include <fcntl.h>
int open(char *fname, int fmode, int p);

■ open Function

[Explanation]

In the mode specified by fmode, open the file having the name specified by fname. For fmode specifying, a combination of the following flags (logical OR) is used. The value "0777" is always delivered as p.

- O_RDONLY: Opens a read-only file.
- O_WRONLY: Opens a write-only file.
- O_RDWR: Opens a read/write file.

The above three flags are to be exclusively specified.

- O_CREAT: Create this flag when the specified file does not exist. If the specified file already exists, ignore this flag.
- O_TRUNC: If any data remains in the file, discard such data to empty the file.
- O_APPEND: Selects the append mode for file opening. The file position prevailing at the time of opening must be set so as to indicate the end of the file. When writing into a file placed in this mode, start writing at the end of the file without regard to the current file position.
- O_BINARY: Specifies a binary file. Therefore, the file opened must be treated as a binary file. Files for which this is not specified must be treated as text files.

When the name for standard input/output and standard error output, which is determined for system environment setup, is specified as the file name for the first argument, allocate the standard input/output and standard error output to the file to be opened.

[Return value]

When file opening is successfully done, the file number must be returned. If file opening is not successfully done, on the other hand, the value "-1" must be returned.

8.5.2 close Function

The close function should be generated according to the specifications in this section.
int close(int fileno);

■ close Function

[Explanation]

The closing process must be performed for the file specified by fileno.

[Return value]

When file closing is successfully done, the value "0" must be returned. If file closing is not successfully done, the value "-1" must be returned.

8.5.3 read Function

The read function should be generated according to the specifications in this section.
int read(int fileno, char *buf, int size);

■ read Function

[Explanation]

From the file specified by fileno, size-byte data must be input into the area specified by buf.

If the text file new line character is other than \n in the system environment at this time, perform setup with the new line character converted to \n by the read function.

[Return value]

When the input from the file is successfully done, the input character count must be returned. If the input from the file is not successfully done, the value -1 must be returned. If the file ends in the middle of the input sequence, a value smaller than size can be returned as the input character count.

8.5.4 write Function

The write function should be generated according to the specifications in this section.
int write (int fileno, char *buf, int size);

■ write Function

[Explanation]

To the file specified by fileno, size-byte data in the area specified by buf must be outputted. If the file is opened in the append mode, the output must always be appended to the end of the file. If the text file new line character is other than \n in the system environment at this time, the output must be generated with the system environment new line character converted to \n by the write function.

[Return value]

When the output to the file is successfully done, the output character count must be returned. If it is not successfully done, the value "-1" must be returned.

8.5.5 lseek Function

The lseek function should be generated according to the specifications in this section.
#include <unistd.h>
long int lseek(int fileno, off_t offset, int whence);

■ lseek Function

[Explanation]

The file specified by fileno must be moved to a position that is offset bytes away from the position specified by whence. The file position is determined according to the byte count from the beginning of the file. The following three positions are to be specified by whence.

- SEEK_CUR: Adds the offset value to the current file position.
- SEEK_END: Adds the offset value to the end of the file.
- SEEK_SET: Adds the offset value to the beginning of the file.

[Return value]

When the file position is successfully changed, the new file position must be returned. If it is not successfully changed, -1L must be returned.

8.5.6 isatty Function

The isatty function should be generated according to the specifications in this section.
int isatty(int fileno);

■ isatty Function

[Explanation]

The file specified by fileno is to be checked to see whether it is a terminal file. When the file is a terminal file, true must be returned. If not, false must be returned.

[Return value]

When the specified file is a terminal file, true must be returned. If not, false must be returned.

8.5.7 sbrk Function

The sbrk function should be generated according to the specifications in this section.
char *sbrk(int size);

■ sbrk Function

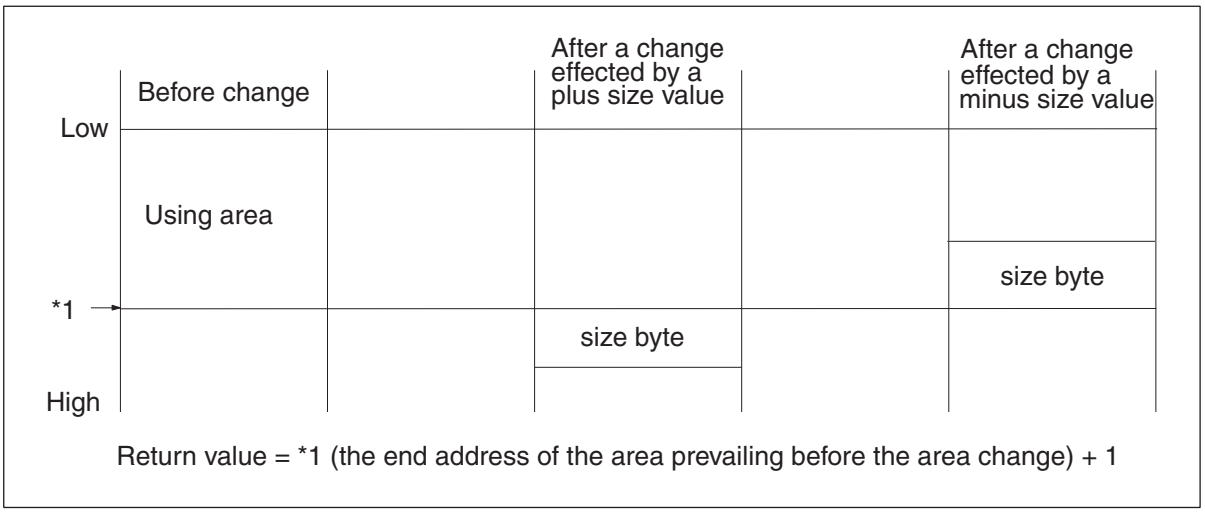
[Explanation]

The existing area must be enlarged by size bytes. If size is a negative quantity, the area must be reduced.

If the sbrk function has not been called, furnish a size-byte area.

The area varies as shown in Figure 8.5-1 by calling the sbrk function.

Figure 8.5-1 Area Change Brought About by sbrk Function Calling



[Return value]

When the area change is successfully made, the value to be returned must be determined by adding the value "1" to the end address of the area prevailing before the area change. If the sbrk function has not been called, the start address of the acquired area must be returned. If the area change is not successfully made, the value (char)-1 must be returned.

8.5.8 _exit Function

The _exit function should be generated according to the specifications in this section.

```
#include  <stdlib.h>
void  _exit (int  status);
```

■ _exit Function

[Explanation]

The _exit function must bring the program to a normal end. When the status value is 0 or in the case of EXIT_SUCCESS, the successful end state must be returned to the system environment. In the case of EXIT_FAILURE, the unsuccessful end state must be returned to the system environment.

[Return value]

The _exit function does not return to the caller.

8.5.9 _abort Function

The _abort function should be generated according to the specifications in this section.
void _abort(void);

■ _abort Function

[Explanation]

The _abort function must bring the program to an abnormal end.

[Return value]

The _abort function does not return to the caller.

8.6 Time Function Specifications

In the time functions, there are clock function to get the processor time used and time function to get current calendar time.
The clock function and the time function must be created to suit the system.

■ Time Function

Create the time functions in compliance with the specifications stated in this section.

8.6.1 clock Function

Create the clock function in compliance with the specifications stated in this section.
#include <time.h>
clock_t clock (void);

■ clock Function

[Explanation]

Please return the processor time used by the program.

Please adjust the return value to become seconds if it is divided by the value of macro CLOCKS_PER_SEC.

[Return value]

When the processor time used is not available or its value cannot be represented, the function returns the value (clock_t)-1.

8.6.2 time Function

Create the time function in compliance with the specifications stated in this section.
#include <time.h>
time_t time (timet *timer);

■ time Function

[Explanation]

Please return the seconds of the current calendar time from "January 1, 1970 00:00:00(UTC)".

[Return value]

When the current calendar time is not available, the return value (time_t)-1.

If timer is not null pointer, the return value is also assigned to the object it points to.

CHAPTER 9

COMPILER-DEPENDENT SPECIFICATIONS

This chapter describes the specifications that vary with the compiler. Descriptions are related to JIS standard that are created based on ANSI standard.

9.1 Compiler-dependent C Language Specification Differentials

Table 9.1-1 lists the compiler-dependent C language specification differentials.

■ Compiler-dependent Language Specification Differentials

Table 9.1-1 Compiler-dependent Language Specification Differentials (1 / 2)

Specification Differentials	Related Section in the JIS Standard	This Compiler
Japanese language process support and code system	5.2.1 "Character sets" 6.1.2 "Identifiers"	No support EUC or SJIS entries can be made in the comment and string literals (cannot be mixed)
Keyword	6.1.1 "Keyword"	Add "_Bool"
Recognized character count of an identifier with an external binding	6.1.2 "Identifiers"	1024
Differentiation between upper- and lower-case alphabetical characters of an identifier with an external binding	6.1.2 "Identifiers"	Treated as different characters
Character set element expression code system	6.1.3 "Numerical constants"	ASCII code
char type treatment and expressible value range	6.2.1.1 "Character type and integer type"	Unsigned (*1) 0 to 255
Floating-point data formats and sizes float type double/long double type	6.1.2.5 "Data types"	IEEE type (*2) 4 bytes 8 bytes
Whether or not to treat the start bit as signed bit when following types specified as bit field char, short, int, and long type	6.5.2.1, "Structure specifier and union specifier"	Not treated as a sign (*1)
Types that can be specified as bit field	6.5.2.1, "Structure specifier and union specifier"	_Bool type char type signed char type unsigned char type short type unsigned short type int type unsigned int type long type unsigned long type

Table 9.1-1 Compiler-dependent Language Specification Differentials (2 / 2)

Specification Differentials	Related Section in the JIS Standard	This Compiler
Structure or union member boundary alignment value _Bool type char type signed char type unsigned char type short int type unsigned short int type int type unsigned int type long int type unsigned long int type float type double type long double type pointer type	6.5.2.1, "Structure specifier and union specifier"	 1 byte 1 byte 1 byte 1 byte 2 bytes 2 bytes 4 bytes 4 bytes 4 bytes 4 bytes 4 bytes 8 bytes 8 bytes 4 bytes
Character constant expression code system for preprocessor	6.8.1 "Conditional Include"	ASCII code
Registers that can be specified within asm statement		R0, R1, R2, R3, R12, R13 (*3)
ANSI-compliant standard library function support		Refer to the APPENDIX A.

*1: Alterable through option use.

*2: See section "9.2 Type of Floating-point Data and Range of Representable Values".

*3: The other registers can be used when they are saved and recovered by the user.

9.2 Type of Floating-point Data and Range of Representable Values

Table 9.2-1 lists the floating-point data types and the range of values that can be expressed for each type.

■ Type of Floating-point Data and Range of Representable Values

Table 9.2-1 Type of Floating-point Data and Range of Representable Values

Type of floating-point data	Range of Representable Values
float type	The exponent part is a value between − 126 and + 127. (Base 2) The fractional portion of the mantissa (the integer portion is normalized to 1) is binary and has 24-digit accuracy.
double type	The exponent part is a value between − 1022 and + 1023. (Base 2) The fractional part of the mantissa (the integer part is normalized to 1) is binary and has 53-digit accuracy.
long double type	The exponent part is value between − 1022 and + 1023. (Base 2) The fractional part of the mantissa (the integer part is normalized to 1) is binary and has 53-digit accuracy.

9.3 Floating-point Operation due to the Runtime Library Function

All floating-point operations, except for ones calculated in the compilation time, are done by the runtime library functions. Although those functions are designed referring to ANSI/IEEE Std754-1985, they do not completely conform to it.
This section describes the differences between the specification of the floating-point runtime library functions and ANSI/IEEE Std754-1985.

■ Arithmetic operation (addition, subtraction, multiplication, and division)

- Rounding of the resultant mantissa part

  Round-to-nearest mode, only.

- Denormalized number

  If the left operand is a denormalized number, it is assumed to be zero with the same sign. If the right operand is a denormalized number, it is assumed to be zero with the same sign, too. In some cases, the denormalized number with the correct sign is returned rather than the strict zero.

- Resultant value under the underflow exception

  It is assumed that the underflow exception occurs when the exponent value of true operation result is too small to be represented as the normalized number. In that case, zero with the correct sign is returned.

- Resultant value under the overflow exception

  Infinity with the correct sign is returned.

- Resultant value under the invalid operation exception

  NaN (Not a number) is returned. In the floating-point runtime library, any routines do not distinguish SNaN (Signaling NaN) and QNaN (Quiet NaN).

- Interrupt at operation exception.

  No interrupt occur.

- Status flag

  Not supported.

■ Comparison

- Denormalized number

  The denormalized number is treated as zero with the same sign.

- Comparison result under the invalid operation exception

  The library function returns uncertain result.

- Interrupt at operation exception

  No interrupt occur.

- Status flag

  Not supported.

■ Type conversion (integer -> floating-point number)

- Rounding of the resultant mantissa part

  Round-to-nearest mode, only.

- Interrupt at operation exception

  No interrupt occur.

- Status flag

  Not supported.

■ Type conversion (floating-point number -> integer)

- Resultant value under the invalid operation exception

  Uncertain value is returned.

- Interrupt at operation exception

  No interrupt occur.

- Status flag

  Not supported.

■ Type conversion (floating-point number -> floating-point number)

- Rounding of the resultant mantissa part

  Round-to-nearest mode, only.

- Denormalized number

  If the converting value is a denormalized number, it is treated as zero with the same sign. In some cases, the denormalized number is returned rather than the strict zero.

- Resultant value under the underflow exception

  It is assumed that the underflow exception occurs when the exponent value of true operation result is too small to be represented as the normalized number. In that case, zero with the correct sign is returned.

- Resultant value under the overflow exception

  Infinity with the correct sign is returned.

- Resultant value under the invalid operation exception

  NaN (Not a Number) is returned. In the floating-point runtime library, any routines do not distinguish SNaN (Signaling NaN) and QNaN (Quiet NaN).

- Interrupt at operation exception

  No interrupt occur.

- Status flag

  Not supported.

9.4 Dissimilarities between C++ Specifications for C/C++ Compiler and ISO

This section explains the dissimilarities between the C++ specifications for the C/C++ compiler and ISO/IEC 14882:1998.

■ Modifications to C++ Specifications for C/C++ Compiler from ISO

Differences between the C++ specifications for the C/C++ compiler and ISO/IEC 14882:1998 are explained below.

- Data in enumerated type cannot take the values exceeding the int type. The values exceeding the int type values, if specified, would cause the diagnosis as warning and the data would be converted into the int type data.
- There is no support for the name-solving function in [temp.res] (Section 14.6) and [temp.dep] (Section 14.6.2) mentioned in the ISO/IEC 14882:1998 Specification.
- There is no support for the template argument function in [temp.arg.template] (Section 14.3.3) mentioned in the ISO/IEC 14882:1998 Specification.
- There is no support for the international character set function such as \uabcd.
- There is no support for the export keyword and export function.
- There is no support for the runtime type identification function including dynamic_cast and typeid operators.
- There is no support for the exception handling function including try-catch and throw.
- Only the C standard and EC++ libraries are provided. Libraries containing templates such as STL are not available.

9.5 C++ Specifications for C/C++ Compiler and EC++ Specifications

This section describes the C++ specifications for the C/C++ compiler and the EC++ specifications.

■ C++ Specifications for C/C++ Compiler and EC++ Specifications

At default, the C/C++ compiler operates to the EC++ specifications. A warning message is output at description of specifications outside the EC++ range. To delete just this warning, specify the -Ja option (including extended specifications) or the -Jc option (stringent specifications).

For details of the EC++ specifications, refer to http://www.caravan.net/ec2plus/.

9.6 Limitations on Use of C++ Template

This section deals with the limitations on use of the C++ template.

■ Limitations on Use of C++ Template

● Options specified when using template

Specify the -Ja option or the -Jc option. If this option is not given, description of specifications is outside the range of the EC++ specifications and the warning message is output.

● Limitations

When instantiation is not controlled by #pragma, or anything other than "local" (default: "none") is selected in the -t option, the following limitations apply:

- Linking is impossible just with the generated object file. The source files and simultaneously generated files with .ii extensions are needed in pairs.
- Do not move, delete or rewrite the generated files with .ii extensions.
- Do not change the directories and names of the generated object and source files and do not link object files after deletion. In this case, always perform recompiling. Recompiling is needed for changes alone between the UNIX and Windows environments.
- When the -S option is specified, always generate object files with .obj extensions with the same module name in the same directories as those of the generated assembler files.
- Data in the assembler files generated by the -S option is not rewritten.
- Do not generate libraries with a librarian.
- For development by more than one operator, carry out tasks from compiling to linking at a single point. Compiling at multiple points requires conformity in the positions of directories of source and object files (including the name of a network drive when used), the environment variables related to the C/C++ compiler, and the directory where the C/C++ compiler is installed.
- When compiling at multiple points in the UNIX environment, the true directory name not including a symbolic link must be the same.
- Linking is delayed depending on how to use a template.
- A warning of errors in the template syntax is issued at linking.

■ Circumventing limitations on the use of the C++ template

To circumvent the above problems, simply specify the -t local option for all the modules to perform compilation. This specification causes template data in each module, resulting in an increased code size for the runtime object.

To solve the code size problem, specify the --no_auto_instantiation option to embed #pragma in the source program, thereby controlling template instantiation.

For information on using #pragma to control instantiation, see section "5.11 Function for Controlling Instantiation of C++ Template".

CHAPTER 10

SIMULATOR DEBUGGER LOW-LEVEL FUNCTION LIBRARY

The simulator debugger low-level function library is a library of the low-level functions which are necessary when the standard library is used with the simulator debugger.
This chapter describes how to use the simulator debugger low-level function library.

10.1 Low-level Function Library Overview

The low-level function library is outlined below.

■ Low-level Function Library Overview

The low-level function library offers the functions that are necessary when the standard library is used with the simulator debugger. The main functions are as follows.

- File manipulation functions based on I/O port simulation (open, close, read, write, lseek, isatty)
- Dynamic memory allocation function (sbrk)

In the simulator debugger, the program executed cannot terminate its own execution. Therefore, prepare the _abort and _exit functions.

■ File System Overview

The low-level function library uses the I/O port simulation function of the simulator debugger to carry out standard input/output operations and input/output operations relative to files. These operations are completed by performing input/output operations relative to one I/O port area which is regarded as one file.

When the open function is called, it allocates a 1-byte area of the I/O port simulation area (I/O section) defined by the low-level function library, and returns as the file number the offset from the beginning of the allocated area.

The read function and write function perform input/output operations relative to the 1-byte area allocated by the open function.

Input/output operations can be performed relative to the standard input/output and files when such standard input/output and files are allocated to the above-mentioned area prior to program execution using simulator debugger commands set inport and set outport.

The close function frees an already allocated area to render it reusable. Since the file position cannot be changed in the simulator debugger, the value "-1" is always returned for the lseek function.

■ Area Management

An already acquired external variable area is used as the area returned by the sbrk function.

When the sbrk function is called, area allocation begins with the lowest address of the area.

10.2 Low-level Function Library Use

This section describes the load module creation and simulator debugger setup procedures to be performed for low-level function library use.

■ Initialization

No initialization is required except for _stream_init function calling.

When creating the startup routine in accordance with the system, call the _stream_init function prior to main function calling.

■ Load Module Creation

After completing creating of the necessary program, compile and link all the necessary modules. No special option specifying is needed.

The following libraries and startup routine are linked.

- startup.obj
- Standard library (lib911.lib, lib911e.lib, lib911p.lib)
- Low-level function library (lib911ie.lib)

The sections are arranged at the following addresses.

- IOPORT: Address 0
- STACK: Address 0x100000
- Other: Address 0x1000

To change the IOPORT section arrangement, specify the -sc IOPORT=address option at compiling. Describe the section arrangement address at the address position.

■ Simulator Debugger Setup

[Setup for Standard Input/Output Use]

```
set inport/ascii IOPORT, 0xff, $TERMINAL
set outport/ascii IOPORT+1, 0xff, $TERMINAL
```

Enter the address where the IOPORT section was positioned at linking in the above IOPORT position. If the -sc option is not specified at linking, the following results.

```
set inport/ascii 0, 0xff, $TERMINAL
set outport/ascii 1,  0xff, $TERMINAL
```

Since the first three areas of the IOPORT section are used for standard input, standard output, and standard error output, the other files are allocated to the fourth and subsequent areas (The offset from the beginning of the IOPORT section is 3).

In other words, allocation is performed sequentially in the order of file opening (offset 3, offset 4, etc.). Therefore, perform setup accordingly using the set inport and set outport commands.

To open a.doc as the input file and then open b.doc as the output file, setup as indicated below.

```
set inport/ascii IOPORT+3, 0xff, "a.doc"
set outport/ascii IOPORT+4, 0xff, "b.doc"
```

<Example>

Create a program that displays the character string "Hello!!" and initiate execution with the simulator debugger.

```
main( )
{
printf("Hello!!\n");
}
```

Create a C-source file named test.c as indicated above.

Compile using the following command.

```
> fcc911s test.c -cpu MB91F154 -l lib911if.lib
```

At completion of the preceding step, test.abs is created. Execute the created file with the simulator debugger.

After startup, input following commands.

```
> set inport/ascii 0x0, 0xff, $TERMINAL
> set outport/ascii 0x1, 0xff, $TERMINAL
> go, end
```

Since standard input is not involved in the above example, the set inport command can be omitted.

10.3 Low-level Func. Function

This section describes the function specific to the simulator debugger low-level functions.

■ Special I/O Port

As far as the low-level functions are concerned, the first three bytes of the I/O section are specified to function as the standard input, standard output, and standard error output, respectively. For such bytes, files No. 0, 1, and 2 are allocated. They are initialized to the opened state.

Table 10.3-1 shows the predefined I/O port.

Table 10.3-1 Predefined I/O Port

Address	File Number	File Type
IOPORT	0	Standard input
IOPORT+1	1	Standard output
IOPORT+2	2	Standard error output

The input from the standard input (file No. 0) is output to the standard output (file No. 1). The input to the standard input (file No. 0) is discontinued if the new line character \n is entered. However, when the input is fed from some other port, the input continues until the required number of characters are read.

■ open Function

The open function finds an unused I/O port area and then returns as the file number the area's offset from the beginning of the I/O section. In such an instance, the file name and open mode are not to be specified. Even if files are opened using the same file name, differing file numbers are assigned to them. Files No. 0, 1, and 2 are initialized to the opened state. Therefore, the open function begins allocation with file No. 3 unless files 0, 1, and 2 are subjected to the close process.

■ read Function

The read function reads data from the I/O port area specified by the address which is determined by adding the specified file number to the I/O section start address. The input from file No. 0 is treated as a line input. When the new line character \n is entered, the read function terminates even if the required character count is not reached. Further, this input is output to the standard output (file No. 1). The input from a file numbered other than 0 is treated as a block input. Reading continues until the required character count is reached.

■ write Function

The write function writes data to the I/O port area specified by the address which is determined by adding the specified file number to the I/O section start address. Unlike the input, the operation does not vary with the I/O port area address

■ lseek Function

The file position cannot be specified in the simulator debugger. Therefore, the value -1, which indicates an unsuccessful file position change, is always returned.

■ isatty Function

In the case of file No. 0, 1, or 2, true is returned. In the other cases, false is returned.

■ close Function

The close function releases the port related to the specified file number.

■ sbrk Function

The simulator debugger does not provide a means of dynamic memory allocation. Therefore, the sbrk function acquires a fixed area and uses it. To change the area or its size, create an alternative function and substitute it for the sbrk function with a librarian. For details, see section "10.4 Low-level Function Library Change".

10.4 Low-level Function Library Change

This section shows how to change the dynamic allocation area.

■ fcc911s Command Source Program List of sbrk Function

The source program required for changing the dynamic area is shown below. The file name must be __STD_LIB_sbrk.c.

```
#define  HEEP_SIZE     16*1024
static   long        brk_siz = 0;
#if    HEEP_SIZE
typedef  int         _heep_t;
#define ROUNDUP(s)   (((s)+sizeof(_heep_t)-1)&~(sizeof(_heep_t)-1))
static   _heep_t     _heep[ROUNDUP(HEEP_SIZE)/sizeof(_heep_t)];
#define              _heep_size       ROUNDUP(HEEP_SIZE)
#else
extern  char         *_heep;
extern  long         _heep_size;
#endif

extern  char   *sbrk(int size)
{
   if (brk_siz + size > _heep_size || brk_siz + size < 0)
        return((char*)-1);
   brk_siz += size;
   return( (char *)_heep + brk_siz - size);
}
```

■ Dynamic Allocation Area Change

Locate the following line in the source program list of sbrk function. Change the value in this line to the dynamic allocation area size (in bytes).

```
#define  HEEP  SIZE    16*1024
```

Use the following commands to compile and update the library.

```
#define  HEEP_SIZE     16*1024
> fcc911s  -O  -c  __STD_LIB_sbrk.c  -cpu  MB91F154
> flibs  -r  __STD_LIB_sbrk.obj  lib911if.lib  -cpu  MB91F154
```

When the above change is made, the dynamic allocation area is secured as the __STD_LIB_sbrk.c static external variable without being positioned at the beginning of the stack.

APPENDIX

The appendix gives a list of types, macros, functions and variables provided by the libraries and describes the operations specific to the libraries (The APPENDIX A and APPENDIX B).
The list of the error message is described (The APPENDIX C).
The list of the reserved pragma directive is described (The APPENDIX D).
Reentrancy of C library function is described (The APPENDIX E).

APPENDIX A List of Types, Macros, Functions, and Variables Provided by C Libraries

The types, macros, functions, and variables provided by the C libraries are listed below.

■ assert.h

● Function

assert

■ ctype.h

● Macros

isalnum	isalpha	iscntrl	isdigit	isgraph
islower	isprint	ispunct	isspace	isupper
isxdigit	tolower	toupper		

■ errno.h

● Macros

EDOM ERANGE

● Variable

errno

■ float.h

- Macros

FLT_RADIX	FLT_ROUNDS	FLT_MANT_DIG	DBL_MANT_DIG
LDBL_NANT_DIG	FLT_DIG	DBL_DIG	LDBL_DIG
FLT_MIN_EXP	DBL_MIN_EXP	LDBL_MIN_EXP	FLT_MIN_10_EXP
DBL_MIN_10_EXP	LDBL_MIN_10_EXP	FLT_MAX_EXP	DBL_MAX_EXP
LDBL_MAX_EXP	FLT_MAX_10_EXP	DBL_MAX_10_EXP	LDBL_MAX_10_EXP
FLT_MAX	DBL_MAX	LDBL_MAX	FLT_EPSILON
DBL_EPSILON	LDBL_EPSILON	FLT_MIN	DBL_MIN
LDBL_MIN			

■ limits.h

- Macros

MB_LEN_MAX	CHAR_BIT	SCHAR_MIN	SCHAR_MAX	UCHAR_MAX
CHAR_MIN	CHAR_MAX	INT_MIN	INT_MAX	UINT_MAX
SHRT_MIN	SHRT_MAX	USHRT_MAX	LONG_MIN	LONG_MAX
ULONG_MAX	LONG_LONG_MIN	LONG_LONG_MAX	ULONG_LONG_MAX	LLONG_MIN
LLONG_MAX	ULLONG_MAX			

■ math.h

- Macros

  HUGE_VAL EDOM ERANGE

- Function

acos	asin	atan	atan2	cos
sin	tan	cosh	sinh	tanh
exp	frexp	ldexp	log	log10
modf	pow	sqrt	ceil	fabs
floor	fmod			

■ setjmp.h

- Type

  jmp_buf

- Macros

  setjmp

- Function

  longjmp

■ stdarg.h

- Type

  va_list

- Macros

  va_start va_arg va_end

■ stddef.h

- Type

  ptrdiff_t size_t

- Macros

NULL offsetof

■ stdio.h

- Type

ptrdiff_t size_t FILE fpos_t

- Macros

NULL	EOF	SEEK_SET	SEEK_CUR	SEEK_END
_IONBF	_IOLBF	_IOFBF	BUFSIZ	stdin
stdout	stderr	putchar	putc	getchar
getc	offsetof			

- Function

putchar	putc	getchar	getc	fclose
fflush	fopen	freopen	setbuf	setvbuf
fprintf	fscanf	printf	scanf	sprintf
sscanf	vfprintf	vprintf	vsprintf	fgetc
fgets	fputc	fputs	gets	puts
ungetc	fread	fwrite	fgetpos	fseek
fsetpos	ftell	rewind	clearerr	feof
ferror				

■ stdlib.h

- Type

ptrdiff_t size_t div_t ldiv_t

- Macros

NULL offsetof EXIT_FAILURE EXIT_SUCCESS RAND_MAX

- Function

atof	atoi	atol	strtod	strtol
strtoul	rand	srand	calloc	free
malloc	realloc	abort	atexit	exit
bsearch	qsort	abs	div	labs
ldiv				

■ string.h

- Type

ptrdiff_t	size_t

- Macros

NULL	offsetof

- Function

memcpy	memmove	strcpy	strncpy	strcat
strncat	memcmp	strcmp	strncmp	memchr
strchr	strcspn	strpbrk	strrchr	strspn
strstr	strtok	memset	strlen	

■ fcntl.h

- Macros

O_RDONLY	O_WRONLY	O_RDWR	O_APPEND	O_CREAT
O_TRUNC	O_BINARY			

■ unistd.h

- Macros

SEEK_SET	SEEK_CUR	SEEK_END

■ sys/types.h

- Type

  off_t

■ time.h

- Type

  clock_t time_t struct tm

- Macros

  CLOCKS_PER_SEC

- Variable

  _TZ

- Function

  asctime() ctime() difftime() gmtime() localtime()
  mktime() strftime()

■ stdbool.h

- Macros

  bool false true __bool_true_false_are_defined

APPENDIX B Operations Specific to C Libraries

The operations specific to the C libraries are described below.

■ Operations Specific to C Libraries

● Diagnostic information printed out by the assert function and assert function termination operation

[Diagnostic information]

Diagnostic information is output to stdout in the following form.

```
< Program Diagnosis *** information of fail expression >
file       : File name expanded by __FILE__
line       : Line number expanded by __LINE__
expression: Expression
```

[Termination operation]

Same as the abort function calling.

● Inspection character sets for isalnum, isalpha, iscntrl, islower, isprint, and isupper functions

- isalnum: '0' to '9', 'a' to 'z', or 'A' to 'Z'
- isalpha: 'a' to 'z' or 'A' to 'Z'
- iscntrl: '\100' to '\037', or '\177'
- islower: 'a' to 'z'
- isprint: '\040' to '\176'
- isupper: 'A' to 'Z'

● Mathematical function return value upon domain error occurrence

- q N a N

● Whether the mathematical function sets up the macro ERANGE value for errno upon underflow condition occurrence

- ERANGE
- The detectable result value must be +0 or -0.
- The undetectable result value is undefined. It depends on the function.

● When the second argument for the fmod function is 0, the domain error must occur or the value 0 must be returned

- The domain error must occur.

● File buffering characteristics

[Input file buffering characteristics]

- IOLBF, IOFBF: Full buffering
- IONBF: No buffering

[Output file buffering characteristics]

- IOFBF: Full buffering
- IOLBF: Line buffering
- IONBF: No buffering

[Full buffering]

Buffering is conducted using all the preset buffer areas. When the input function is called at the time of input from a file, any data remaining in the buffer is returned as the input from the file. If the buffer is emptied of data or does not have sufficient data, the input from the file is received until the buffer is filled up and then only the necessary amount is returned as the input. At the time of output to a file, the output function writes into the buffer instead of outputting into the file. When the buffer is filled up by the write operation, the buffer outputs its entire contents to the file.

[Line buffering]

Buffering is conducted for each output line.

[No buffering]

File input/output is implemented in compliance with the input/output request made by input/output function calling. Unlike the other buffering operations, no data will be saved into the memory.

● %p format conversion output format for fprintf function

- The number of digits is less than 8 at 8-digit hexadecimal notation, preceding 0s are padded. Uppercase alphabetical characters are used.

● %p format conversion input format for fscanf function

If less then 8 digits in hexadecimal notation are used (in upper- or lower-case alphabetic characters), the leading 0s are padded. If the specified number of digits is exceeded, only the low-order part is valid.

● Length of string literal that can be treated by %s format conversion in printf function, fprintf function, vprintf function and vprintf function

512 characters (value of BUFSIZ macro defined in stdio.h)

● Interpretation of a single "-" character appearing at a position other than the start and end of the scan-list relative to %[format conversion]

A string of consecutive characters beginning with the character placed to the left of "-" and ending with the character placed to the right of "-" is handled.

[Example]

%[a-c] is equal to [abc].

● abort function operation relative to an open file

Closing takes place after flushing of all streams.

● Status returned by the exit function when the argument value is other than 0, EXIT_SUCCESS, and EXIT_FAILURE

The status to be returned is the same as for EXIT_FAILURE.

- Floating-point number limit values

```
FLT_MAX       7F7F  FFFF
DBL_MAX       7FEF  FFFF  FFFF  FFFF
FLT_EPSILON   3400  0000
DBL_EPSILON   3CB0  0000  0000  0000
FLT_MIN       0080  0000
DBL_MIN       0010  0000  0000  0000
```

- Limitations on setjmp function and longjmp function

  The interrupt environment is not supported by the libraries. Therefore, the interrupt handler cannot achieve environment saving and the return to the interrupt handler cannot be made.

- Limitations on va_start macro

  Do not use the following variable definitions for va_start macro second argument.

  - char type, unsigned char type, short type, or unsigned short type (however, the pointer type for these types can be used)
  - Type having the register storage area class
  - Function type
  - Array type
  - Type different from the type derived from existing argument extension

- File types

  Files that can be handled by the libraries are divided into two types; text files and binary files. The libraries treat the text files and binary files in the same manner except for the difference in the second argument of the open function called upon file opening. When a binary file is specified, O_BINARY is added to the second argument of the open function. For the open function argument, see section "8.5.1 open Function".

- div_t type and ldiv_t type

  It is equivalent to an undermentioned structural body.

```
div_t:   struct {
            int    quot;
            int    rem;
         }:
ldiv_t:  struct    {
            long   int quot;
            long   int rem;
         };
```

- abort function operations

  When the abort function is called, all the open output streams are flushed and then all the open streams are closed. Finally, the _abort function is called.

● Maximum count of functions that can be registered by the atexit function

Up to 32 functions can be registered.

● exit function operations

When the exit function is called, all the functions registered by the atexit function are called in the reverse order of registration, all the open output streams are flushed, and then all the open streams are closed. Finally, the _exit function is called with the status value, which is delivered as the argument, retained. When the status value is 0 or EXIT_SUCCESS, it indicates successful termination. When the status value is EXIT_FAILURE, it indicates the unsuccessful termination.

● about accuracy of mathematical functions

The following 16 functions of mathematical functions declared in standard header file math.h calculate the return value by the approximate calculation. Therefor, enough accuracy might not be obtained.

```
acos    asin    atan    atan2    cos
sin     tan     cosh    sinh     tanh
exp     log     log10   pow      sqrt
fmod
```

● The Date and Time library

When the Date and Time library is used, specify the -K LONGLONG option.

● clock_t type

The clock_t type is defined as long long int type.

Please adjust the value to become seconds if it is divided by the value of macro CLOCKS_PER_SEC.

● time_t type

The clock_t type and the time_t type are defined as long long int type.

The range of the date that can be used in the library is as follows.

The starting point	January 1, 1970 00:00:00(UTC)
Minimum unit	1 second
Maximum years	About 292.4 billion years.

Because the long long int type is used, it is not possible to connect directly with IO.

When a negative value is used to indicate before 1970, the result of the library function is not guaranteed.

About clock function and time function

Because the clock() and the time() are functions that depend on the system, it is not included in the standard library. Please make these functions referring to "8.6 Time Function Specifications".

About time zone

Please set the time zone to global variable _TZ.

Please set 9*3600 for JST (Japan Standard Time) because the unit of the value set to _TZ is "second".

To set _TZ, please include time.h.

Please initialize _TZ when you use the mktime function, the ctime function and the localtime function.

Please do not change the value of _TZ while executing the mktime function, the ctime function, and the localtime function. The result when _TZ is changed is not guaranteed. Please change after executing these functions when you change _TZ.

About the Daylight Saving Time

There is no means to obtain information on whether the Daylight Saving Time is used. Therefore, when using functions which use the value of the Daylight Saving Time flag, it is necessary to correct tm_isdst which is a member of the struct tm type.

The operation of the function that uses the tm_isdst is shown as follows.

- The localtime function sets -1 to the value of tm_isdst.
- The ctime function returns the string literal that does not use the Daylight Saving Time.
- The operation of the mktime function is assumed that tm_isdst is a correct value.
- When the value of tm_isdst is -1, the mktime function doesn't use the Daylight Saving Time.
- The mktime function doesn't modify the value of tm_isdst.

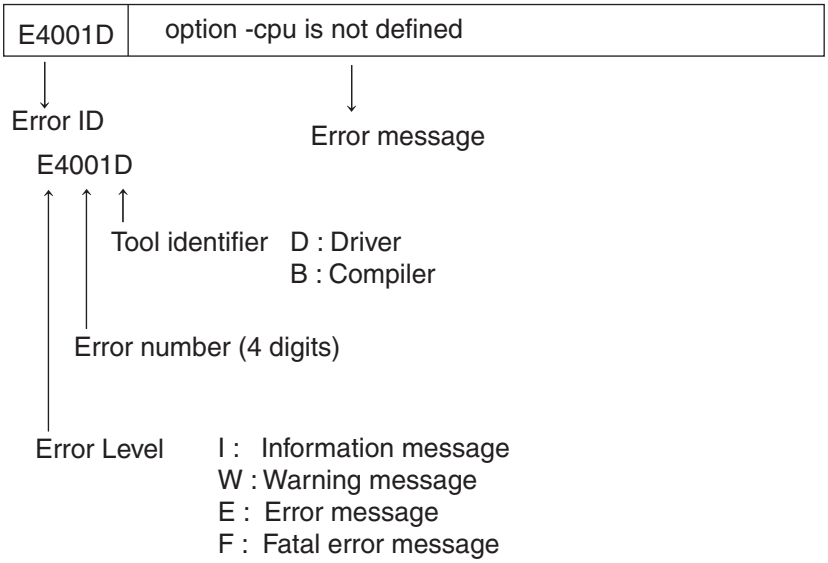
Locale that can be used by strftime function

Only "C" locale.

APPENDIX C Error Message

The compiler displays the error messages below.

■ Format of error messages



W1001D	Not support this option on this version

[Explanation]

Not support this option on this version.

W1002D	Not specified OPT911

[Explanation]

Not specified OPT911.

E4001D	option -cpu is not defined

[Explanation]

Option -cpu is not defined.

E4002D	Not support C++ source file

[Explanation]

Not support C++ source file.

E4003D	illegal cpu name

[Explanation]

Illegal cpu name.

E4004D	CPU information file not found

[Explanation]

CPU information file not found.

E4005D	CPU information not found

[Explanation]

CPU information not found.

E4006D	Not exist file

[Explanation]

Not exist file.

F9001D	source filename not specified

[Explanation]

Source filename not specified.

F9002D	illegal option-name

[Explanation]

Illegal option-name.

F9003D	illegal value

[Explanation]

Illegal value.

F9004D	illegal sub-option

[Explanation]

Illegal sub-option.

F9005D	illegal parameter description

[Explanation]

Illegal parameter description.

F9006D	cannot open option-file

[Explanation]

Cannot open option-file.

F9007D	nested option-file exceeds 8

[Explanation]

Nested option-file exceeds 8.

F9008D	insufficient memory

[Explanation]

Insufficient memory.

F9009D	cannot open file

[Explanation]

Cannot open file.

F9010D	illegal section specification

[Explanation]

Illegal section specification.

F9011D	illegal tool-item

[Explanation]

Illegal tool-item.

F9012D	tool execute is failed

[Explanation]

Tool execute is failed.

F9013D	illegal source file name

[Explanation]

Illegal source file name.

F9014D	internal error

[Explanation]

Internal error.

F9015D	SIGHUP

[Explanation]

Receive SIGHUP signal.

F9016D	SIGINT

[Explanation]

Receive SIGINT signal.

F9017D	SIGBUS

[Explanation]

Receive SIGBUS signal.

F9018D	SIGSEGV

[Explanation]

Receive SIGSEGV signal.

F9019D	SIGTERM

[Explanation]

Receive SIGTERM signal.

F9020D	mismatch CPU information file version

[Explanation]

Mismatch CPU information file version.

F9021D	internal error

[Explanation]

Internal error.

F9022D	insufficient memory

[Explanation]

Insufficient memory.

F9023D	illegal CPU information file format

[Explanation]

Illegal CPU information file format.

I0008B	missing closing quote

[Explanation]

Missing closing quote.

I0010B	"#" not expected here

[Explanation]

"#" not expected here.

I0018B	expected a ")"

[Explanation]

Expected a ")".

I0083B	type qualifier specified more than once

[Explanation]

Type qualifier specified more than once.

I0117B	non-void function "entity" should return a value

[Explanation]

Non-void function "entity" should return a value.

I0142B	expression must have pointer-to-object type

[Explanation]

Expression must have pointer-to-object type.

I0172B	external/internal linkage conflict with previous declaration

[Explanation]

External/internal linkage conflict with previous declaration.

I0177B	entity-kind "entity" was declared but never referenced

[Explanation]

Entity-kind "entity" was declared but never referenced.

I0180B	argument is incompatible with formal parameter

[Explanation]

Argument is incompatible with formal parameter.

I0181B	argument is incompatible with corresponding format string conversion

[Explanation]

Argument is incompatible with corresponding format string conversion.

I0193B	zero used for undefined preprocessing identifier

[Explanation]

Zero used for undefined preprocessing identifier.

I0223B	function declared implicitly

[Explanation]

Function declared implicitly.

I0228B	trailing comma is nonstandard

[Explanation]

Trailing comma is nonstandard.

I0236B	controlling expression is constant

[Explanation]

Controlling expression is constant.

I0237B	selector expression is constant

[Explanation]

Selector expression is constant.

I0260B	explicit type is missing ("int" assumed)

[Explanation]

Explicit type is missing ("int" assumed).

I0261B	access control not specified ("xxxx" by default)

[Explanation]

Access control not specified ("xxxx" by default).

I0324B	duplicate friend declaration

[Explanation]

Duplicate friend declaration.

I0340B	value copied to temporary, reference to temporary used

[Explanation]

Value copied to temporary, reference to temporary used.

I0381B	extra ";" ignored

[Explanation]

Extra ";" ignored.

I0399B	entity-kind "entity" has an operator new xxxx() but no default operator delete xxxx()

[Explanation]

Entity-kind "entity" has an operator new xxxx() but no default operator delete xxxx().

I0400B	entity-kind "entity" has a default operator delete xxxx() but no operator new xxxx()

[Explanation]

Entity-kind "entity" has a default operator delete xxxx() but no operator new xxxx().

I0401B	destructor for base class "type" is not virtual

[Explanation]

Destructor for base class "type" is not virtual.

I0445B	entity-kind "entity" is not used in declaring the parameter types of entity-kind "entity"

[Explanation]

Entity-kind "entity" is not used in declaring the parameter types of entity-kind "entity".

I0451B	omission of "xxxx" is nonstandard

[Explanation]

Omission of "xxxx" is nonstandard.

I0479B	entity-kind "entity" redeclared "inline" after being called

[Explanation]

Entity-kind "entity" redeclared "inline" after being called.

I0487B	inline entity-kind "entity" cannot be explicitly instantiated

[Explanation]

Inline entity-kind "entity" cannot be explicitly instantiated.

I0534B	use of a local type to specify an exception

[Explanation]

Use of a local type to specify an exception.

I0535B	redundant type in exception specification

[Explanation]

Redundant type in exception specification.

I0550B	entity-kind "entity" was set but never used

[Explanation]

Entity-kind "entity" was set but never used.

I0560B	"entity" is reserved for future use as a keyword

[Explanation]

"entity" is reserved for future use as a keyword.

I0650B	calling convention specified here is ignored

[Explanation]

Calling convention specified here is ignored.

I0652B	calling convention is ignored for this type

[Explanation]

Calling convention is ignored for this type.

I0678B	call of entity-kind "entity" (declared at line xxxx) cannot be inlined

[Explanation]

Call of entity-kind "entity" (declared at line xxxx) cannot be inlined.

I0679B	entity-kind "entity" cannot be inlined

[Explanation]

Entity-kind "entity" cannot be inlined.

I0815B	type qualifier on return type is meaningless

[Explanation]

Type qualifier on return type is meaningless.

I0826B	entity-kind "entity" was never referenced

[Explanation]

Entity-kind "entity" was never referenced.

I0831B	support for placement delete is disabled

[Explanation]

Support for placement delete is disabled.

I0837B	omission of explicit type is nonstandard ("int" assumed)

[Explanation]

Omission of explicit type is nonstandard ("int" assumed).

I0863B	effect of this "#pragma pack" directive is local to entity-kind "entity"

[Explanation]

Effect of this "#pragma pack" directive is local to entity-kind "entity".

I0866B	exception specification ignored

[Explanation]

Exception specification ignored.

I0925B	a type qualifier cannot be applied to a function type

[Explanation]

A type qualifier cannot be applied to a function type.

I0938B	return type "int" omitted in declaration of function "main"

[Explanation]

Return type "int" omitted in declaration of function "main".

I0940B	missing return statement at end of non-void "entity"

[Explanation]

Missing return statement at end of non-void "entity".

I0997B	affinity has shared type (not pointer to shared)

[Explanation]

Affinity has shared type, not pointer to shared.

I0998B	affinity expression ignored in nested upc_forall

[Explanation]

Affinity expression ignored in nested upc_forall.

I0999B	bracketed expression is assumed to be a block size specification rather than an array dimension

[Explanation]

Bracketed expression is assumed to be a block size specification rather than an array dimension.

W1001B	last line of file ends without a newline

[Explanation]

Last line of file ends without a newline.

W1002B	last line of file ends with a backslash

[Explanation]

Last line of file ends with a backslash.

W1007B	unrecognized token

[Explanation]

Unrecognized token.

W1009B	nested comment is not allowed

[Explanation]

Nested comment is not allowed.

W1011B	unrecognized preprocessing directive

[Explanation]

Unrecognized preprocessing directive.

W1012B	parsing restarts here after previous syntax error

[Explanation]

Parsing restarts here after previous syntax error.

W1014B	extra text after expected end of preprocessing directive

[Explanation]

Extra text after expected end of preprocessing directive.

W1020B	identifier "xxx" is undefined

[Explanation]

Identifier "xxx" is undefined.

When the -Jc option is not specified, this message is output to the following descriptions.

- A function call without the prototype declaration.
  There should be a prototype declaration for each function call.

When this warning is encountered, the type of the return value and arguments of a function call without the prototype declaration is assumed as follows:

- The type of the return value : int
- Type of arguments : The result of the default argument promotions.

W1021B	type qualifiers are meaningless in this declaration

[Explanation]

Type qualifiers are meaningless in this declaration.

W1022B	invalid hexadecimal number

[Explanation]

Invalid hexadecimal number.

W1023B	integer constant is too large

[Explanation]

Integer constant is too large.

W1024B	invalid octal digit

[Explanation]

Invalid octal digit.

W1026B	too many characters in character constant

[Explanation]

Too many characters in character constant.

W1027B	character value is out of range

[Explanation]

Character value is out of range.

W1030B	floating constant is out of range

[Explanation]

Floating constant is out of range.

W1032B	expression must have arithmetic type

[Explanation]

Expression must have arithmetic type.

W1038B	directive is not allowed - an #else has already appeared

[Explanation]

Directive is not allowed. An #else has already appeared.

W1040B	expected an identifier

[Explanation]

Expected an identifier.

W1042B	operand types are incompatible ("type" and "type")

[Explanation]

Operand types are incompatible ("type" and "type").

W1045B	#undef may not be used on this predefined name

[Explanation]

#undef may not be used on this predefined name.

W1046B	this predefined name may not be redefined

[Explanation]

This predefined name may not be redefined.

W1047B	incompatible redefinition of macro "entity" (declared at line xxxx)

[Explanation]

Incompatible redefinition of macro "entity" (declared at line xxxx).

W1054B	too few arguments in macro invocation

[Explanation]

Too few arguments in macro invocation.

W1055B	too many arguments in macro invocation

[Explanation]

Too many arguments in macro invocation.

W1061B	integer operation result is out of range

[Explanation]

Integer operation result is out of range.

W1062B	shift count is negative

[Explanation]

Shift count is negative.

W1063B	shift count is too large

[Explanation]

Shift count is too large.

W1064B	declaration does not declare anything

[Explanation]

Declaration does not declare anything.

W1065B	expected a ";"

[Explanation]

Expected a ";".

W1066B	enumeration value is out of "int" range

[Explanation]

Enumeration value is out of "int" range.

W1068B	integer conversion resulted in a change of sign

[Explanation]

Integer conversion resulted in a change of sign.

W1069B	integer conversion resulted in truncation

[Explanation]

Integer conversion resulted in truncation.

W1070B	incomplete type is not allowed

[Explanation]

Incomplete type is not allowed.

W1076B	argument to macro is empty

[Explanation]

Argument to macro is empty.

W1077B	this declaration has no storage class or type specifier

[Explanation]

This declaration has no storage class or type specifier.

W1080B	a storage class may not be specified here

[Explanation]

A storage class may not be specified here.

W1082B	storage class is not first

[Explanation]

Storage class is not first.

W1083B	type qualifier specified more than once

[Explanation]

Type qualifier specified more than once.

W1084B	invalid combination of type specifiers

[Explanation]

Invalid combination of type specifiers.

W1085B	invalid storage class for a parameter

[Explanation]

Invalid storage class for a parameter.

W1086B	invalid storage class for a function

[Explanation]

Invalid storage class for a function.

W1096B	a translation unit must contain at least one declaration

[Explanation]

A translation unit must contain at least one declaration.

W1099B	a declaration here must declare a parameter

[Explanation]

A declaration here must declare a parameter.

W1101B	"xxxx" has already been declared in the current scope

[Explanation]

"xxxx" has already been declared in the current scope.

W1102B	forward declaration of enum type is nonstandard

[Explanation]

Forward declaration of enum type is nonstandard.

W1107B	zero-length bit field must be unnamed

[Explanation]

Zero-length bit field must be unnamed.

W1108B	signed bit field of length 1

[Explanation]

Signed bit field of length 1.

W1111B	statement is unreachable

[Explanation]

Statement is unreachable.

W1114B	entity-kind "entity" was referenced but not defined

[Explanation]

Entity-kind "entity" was referenced but not defined.

W1117B	non-void function "entity" should return a value

[Explanation]

Non-void function "entity" should return a value.

W1118B	a void function may not return a value

[Explanation]

A void function may not return a value.

W1120B	return value type does not match the function type

[Explanation]

Return value type does not match the function type.

W1127B	expected a statement

[Explanation]

Expected a statement.

W1128B	loop is not reachable from preceding code

[Explanation]

Loop is not reachable from preceding code.

W1129B	a block-scope function may only have extern storage class

[Explanation]

A block-scope function may only have extern storage class.

W1137B	expression must be a modifiable lvalue

[Explanation]

Expression must be a modifiable lvalue.

W1138B	taking the address of a register variable is not allowed

[Explanation]

Taking the address of a register variable is not allowed.

W1139B	taking the address of a bit field is not allowed

[Explanation]

Taking the address of a bit field is not allowed.

W1140B	too many arguments in function call

[Explanation]

Too many arguments in function call.

W1142B	expression must have pointer-to-object type

[Explanation]

Expression must have pointer-to-object type.

W1144B	a value of type "type" cannot be used to initialize an entity of type "type"

[Explanation]

A value of type "type" cannot be used to initialize an entity of type "type".

W1147B	declaration is incompatible with entity-kind "entity" (declared at line xxxx)

[Explanation]

Declaration is incompatible with entity-kind "entity" (declared at line xxxx).

W1152B	conversion of nonzero integer to pointer

[Explanation]

Conversion of nonzero integer to pointer.

W1155B	old-fashioned assignment operator

[Explanation]

Old-fashioned assignment operator.

W1156B	old-fashioned initializer

[Explanation]

Old-fashioned initializer.

W1159B	declaration is incompatible with previous "entity" (declared at line xxxx)

[Explanation]

Declaration is incompatible with previous "entity" (declared at line xxxx).

W1161B	unrecognized #pragma

[Explanation]

Unrecognized #pragma.

W1165B	too few arguments in function call

[Explanation]

Too few arguments in function call.

W1166B	invalid floating constant

[Explanation]

Invalid floating constant.

W1167B	argument of type "type" is incompatible with parameter of type "type"

[Explanation]

Argument of type "type" is incompatible with parameter of type "type".

W1170B	pointer points outside of underlying object

[Explanation]

Pointer points outside of underlying object.

W1171B	invalid type conversion

[Explanation]

Invalid type conversion.

W1172B	external/internal linkage conflict with previous declaration

[Explanation]

External/internal linkage conflict with previous declaration.

W1173B	floating-point value does not fit in required integral type

[Explanation]

Floating-point value does not fit in required integral type

W1174B	expression has no effect

[Explanation]

Expression has no effect.

W1175B	subscript out of range

[Explanation]

Subscript out of range.

W1177B	entity-kind "entity" was declared but never referenced

[Explanation]

Entity-kind "entity" was declared but never referenced.

W1178B	"&" applied to an array has no effect

[Explanation]

"&" applied to an array has no effect.

W1179B	right operand of "%" is zero

[Explanation]

Right operand of "%" is zero.

W1180B	argument is incompatible with formal parameter

[Explanation]

Argument is incompatible with formal parameter.

W1181B	argument is incompatible with corresponding format string conversion

[Explanation]

Argument is incompatible with corresponding format string conversion.

W1183B	type of cast must be integral

[Explanation]

Type of cast must be integral.

W1185B	dynamic initialization in unreachable code

[Explanation]

Dynamic initialization in unreachable code.

W1186B	pointless comparison of unsigned integer with zero

[Explanation]

Pointless comparison of unsigned integer with zero.

W1187B	use of "=" where "==" may have been intended

[Explanation]

Use of "=" where "==" may have been intended.

W1188B	enumerated type mixed with another type

[Explanation]

Enumerated type mixed with another type.

W1191B	type qualifier is meaningless on cast type

[Explanation]

Type qualifier is meaningless on cast type.

W1192B	unrecognized character escape sequence

[Explanation]

Unrecognized character escape sequence.

W1224B	the format string requires additional arguments

[Explanation]

The format string requires additional arguments.

W1225B	the format string ends before this argument

[Explanation]

The format string ends before this argument.

W1226B	invalid format string conversion

[Explanation]

Invalid format string conversion.

W1228B	trailing comma is nonstandard

[Explanation]

Trailing comma is nonstandard.

W1229B	bit field cannot contain all values of the enumerated type

[Explanation]

Bit field cannot contain all values of the enumerated type.

W1230B	nonstandard type for a bit field

[Explanation]

Nonstandard type for a bit field.

W1231B	declaration is not visible outside of function

[Explanation]

Declaration is not visible outside of function.

W1232B	old-fashioned typedef of "void" ignored

[Explanation]

Old-fashioned typedef of "void" ignored.

W1233B	left operand is not a struct or union containing this field

[Explanation]

Left operand is not a struct or union containing this field.

W1234B	pointer does not point to struct or union containing this field

[Explanation]

Pointer does not point to struct or union containing this field.

W1236B	controlling expression is constant

[Explanation]

Controlling expression is constant.

W1240B	duplicate specifier in declaration

[Explanation]

Duplicate specifier in declaration.

W1257B	const entity-kind "entity" requires an initializer

[Explanation]

Const entity-kind "entity" requires an initializer.

W1260B	explicit type is missing ("int" assumed)

[Explanation]

Explicit type is missing ("int" assumed).

W1267B	old-style parameter list (anachronism)

[Explanation]

Old-style parameter list (anachronism).

W1284B	NULL reference is not allowed

[Explanation]

NULL reference is not allowed.

W1290B	copy constructor for class "type" is ambiguous

[Explanation]

Copy constructor for class "type" is ambiguous.

W1294B	invalid union member -- class "type" has a disallowed member function

[Explanation]

Invalid union member -- class "type" has a disallowed member function.

W1296B	invalid use of non-lvalue array

[Explanation]

Invalid use of non-lvalue array.

W1301B	typedef name has already been declared (with same type)

[Explanation]

Typedef name has already been declared (with same type).

W1310B	default argument of type "type" is incompatible with parameter of type "type"

[Explanation]

Default argument of type "type" is incompatible with parameter of type "type".

W1313B	type qualifier is not allowed on this function

[Explanation]

Type qualifier is not allowed on this function.

W1326B	"inline" is not allowed

[Explanation]

"inline" is not allowed.

W1330B	entity-kind "entity" is inaccessible

[Explanation]

Entity-kind "entity" is inaccessible.

W1334B	class "type" has no suitable copy constructor

[Explanation]

Class "type" has no suitable copy constructor.

W1335B	linkage specification is not allowed

[Explanation]

Linkage specification is not allowed.

W1337B	linkage specification is incompatible with previous "entity" (declared at line xxxx)

[Explanation]

Linkage specification is incompatible with previous "entity" (declared at line xxxx).

W1341B	"operator" xxxx must be a member function

[Explanation]

"operator xxxx" must be a member function.

W1354B	first parameter of deallocation function must be of type "void *"

[Explanation]

First parameter of deallocation function must be of type "void *".

W1358B	base class name required -- "type" assumed (anachronism)

[Explanation]

Base class name required -- "type" assumed (anachronism).

W1361B	assignment to "this" (anachronism)

[Explanation]

Assignment to "this" (anachronism).

W1362B	"overload" keyword used (anachronism)

[Explanation]

"overload" keyword used (anachronism).

W1368B	entity-kind "entity" defines no constructor to initialize the following:

[Explanation]

Entity-kind "entity" defines no constructor to initialize the following.

W1370B	entity-kind "entity" has an uninitialized const field

[Explanation]

Entity-kind "entity" has an uninitialized const field.

W1375B	declaration requires a typedef name

[Explanation]

Declaration requires a typedef name.

W1379B	cast of bound function to normal function pointer (anachronism)

[Explanation]

Cast of bound function to normal function pointer (anachronism).

W1381B	extra ";" ignored

[Explanation]

Extra ";" ignored.

W1382B	nonstandard member constant declaration (standard form is a static const integral member)

[Explanation]

Nonstandard member constant declaration (standard form is a static const integral member).

W1387B	delete array size expression used (anachronism)

[Explanation]

Delete array size expression used (anachronism).

W1395B	single-argument function used for postfix "xxxx" (anachronism)

[Explanation]

Single-argument function used for postfix "xxxx" (anachronism).

W1398B	cast to array type is nonstandard (treated as cast to "type")

[Explanation]

Cast to array type is nonstandard (treated as cast to "type").

W1403B	entity-kind "entity" has already been declared

[Explanation]

Entity-kind "entity" has already been declared.

W1406B	using nested entity-kind "entity" (anachronism)

[Explanation]

Using nested entity-kind "entity" (anachronism).

W1414B	delete of pointer to incomplete class

[Explanation]

Delete of pointer to incomplete class.

W1425B	dollar sign ("$") used in identifier

[Explanation]

Dollar sign ("$") used in identifier.

W1426B	temporary used for initial value of reference to non-const (anachronism)

[Explanation]

Temporary used for initial value of reference to non-const (anachronism).

W1427B	qualified name is not allowed in member declaration

[Explanation]

Qualified name is not allowed in member declaration.

W1428B	enumerated type mixed with another type (anachronism)

[Explanation]

Enumerated type mixed with another type (anachronism).

W1430B	returning reference to local temporary

[Explanation]

Returning reference to local temporary.

W1445B	entity-kind "entity" is not used in declaring the parameter types of entity-kind "entity"

[Explanation]

Entity-kind "entity" is not used in declaring the parameter types of entity-kind "entity".

W1450B	the type "long long" is nonstandard

[Explanation]

The type "long long" is nonstandard.

W1451B	omission of "xxxx" is nonstandard

[Explanation]

Omission of "xxxx" is nonstandard.

W1458B	argument of type "type" is incompatible with template parameter of type "type"

[Explanation]

Argument of type "type" is incompatible with template parameter of type "type".

W1460B	declaration of "xxxx" hides function parameter

[Explanation]

Declaration of "xxxx" hides function parameter.

W1461B	initial value of reference to non-const must be an lvalue

[Explanation]

Initial value of reference to non-const must be an lvalue.

W1469B	tag kind of xxxx is incompatible with declaration of entity-kind "entity" (declared at line xxxx)

[Explanation]

Tag kind of xxxx is incompatible with declaration of entity-kind "entity" (declared at line xxxx).

W1472B	member function typedef (allowed for cfront compatibility)

[Explanation]

Member function typedef (allowed for cfront compatibility).

W1482B	entity-kind "entity" is an inaccessible type (allowed for cfront compatibility)

[Explanation]

Entity-kind "entity" is an inaccessible type (allowed for cfront compatibility).

W1494B	declaring a void parameter list with a typedef is nonstandard

[Explanation]

Declaring a void parameter list with a typedef is nonstandard.

W1495B	global entity-kind "entity" used instead of entity-kind "entity" (cfront compatibility)

[Explanation]

Global entity-kind "entity" used instead of entity-kind "entity" (cfront compatibility).

W1497B	declaration of "xxxx" hides template parameter

[Explanation]

Declaration of "xxxx" hides template parameter.

W1504B	nonstandard form for taking the address of a member function

[Explanation]

Nonstandard form for taking the address of a member function.

W1512B	type qualifier on a reference type is not allowed

[Explanation]

Type qualifier on a reference type is not allowed.

W1513B	a value of type "type" cannot be assigned to an entity of type "type"

[Explanation]

A value of type "type" cannot be assigned to an entity of type "type".

W1514B	pointless comparison of unsigned integer with a negative constant

[Explanation]

Pointless comparison of unsigned integer with a negative constant.

W1516B	const object requires an initializer

[Explanation]

Const object requires an initializer.

W1518B	nonstandard preprocessing directive

[Explanation]

Nonstandard preprocessing directive.

W1520B	initialization with "{...}" expected for aggregate object

[Explanation]

Initialization with "{...}" expected for aggregate object.

W1522B	pointless friend declaration

[Explanation]

Pointless friend declaration.

W1523B	"." used in place of "::" to form a qualified name (cfront anachronism)

[Explanation]

"." used in place of "::" to form a qualified name (cfront anachronism).

W1524B	non-const function called for const object (anachronism)

[Explanation]

Non-const function called for const object (anachronism).

W1533B	handler is potentially masked by previous handler for type "type"

[Explanation]

Handler is potentially masked by previous handler for type "type".

W1541B	omission of exception specification is incompatible with previous entity-kind "entity" (declared at line xxxx)

[Explanation]

Omission of exception specification is incompatible with previous entity-kind "entity" (declared at line xxxx).

W1544B	use of a local type to declare a nonlocal variable

[Explanation]

Use of a local type to declare a nonlocal variable.

W1545B	use of a local type to declare a function

[Explanation]

Use of a local type to declare a function.

W1546B	transfer of control bypasses initialization of:

[Explanation]

Transfer of control bypasses initialization of.

W1549B	entity-kind "entity" is used before its value is set

[Explanation]

Entity-kind "entity" is used before its value is set.

W1550B	entity-kind "entity" was set but never used

[Explanation]

Entity-kind "entity" was set but never used.

W1552B	exception specification is not allowed

[Explanation]

Exception specification is not allowed.

W1553B	external/internal linkage conflict for entity-kind "entity" (declared at line xxxx)

[Explanation]

External/internal linkage conflict for entity-kind "entity" (declared at line xxxx).

W1554B	entity-kind "entity" will not be called for implicit or explicit conversions

[Explanation]

Entity-kind "entity" will not be called for implicit or explicit conversions.

W1560B	"entity" is reserved for future use as a keyword

[Explanation]

"entity" is reserved for future use as a keyword.

W1605B	floating-point template parameter is nonstandard

[Explanation]

Floating-point template parameter is nonstandard.

W1606B	this pragma must immediately precede a declaration

[Explanation]

This pragma must immediately precede a declaration.

W1607B	this pragma must immediately precede a statement

[Explanation]

This pragma must immediately precede a statement.

W1608B	this pragma must immediately precede a declaration or statement

[Explanation]

This pragma must immediately precede a declaration or statement.

W1609B	this kind of pragma may not be used here

[Explanation]

This kind of pragma may not be used here.

W1610B	entity-kind "entity" does not match "entity" -- virtual function override intended?

[Explanation]

Entity-kind "entity" does not match "entity". --virtual function override intended.

W1611B	overloaded virtual function "entity" is only partially overridden in entity-kind "entity"

[Explanation]

Overloaded virtual function "entity" is only partially overridden in entity-kind "entity".

W1617B	pointer-to-member-function cast to pointer to function

[Explanation]

Pointer-to-member-function cast to pointer to function.

W1618B	struct or union declare no named members

[Explanation]

Struct or union declare no named members.

W1619B	nonstandard unnamed field

[Explanation]

Nonstandard unnamed field.

W1620B	nonstandard unnamed member

[Explanation]

Nonstandard unnamed member.

W1626B	precompiled header file "xxxx" is either invalid or not generated by this version of the compiler

[Explanation]

Precompiled header file "xxxx" is either invalid or not generated by this version of the compiler.

W1627B	precompiled header file "xxxx" was not generated in this directory

[Explanation]

Precompiled header file "xxxx" was not generated in this directory.

W1628B	header files used to generate precompiled header file "xxxx" have changed

[Explanation]

Header files used to generate precompiled header file "xxxx" have changed.

W1629B	the command line options do not match those used when precompiled header file "xxxx" was created

[Explanation]

The command line options do not match those used when precompiled header file "xxxx" was created.

W1630B	the initial sequence of preprocessing directives is not compatible with those of precompiled header file "xxxx"

[Explanation]

The initial sequence of preprocessing directives is not compatible with those of precompiled header file "xxxx".

W1634B	memory usage conflict with precompiled header file "xxxx"

[Explanation]

Memory usage conflict with precompiled header file "xxxx".

W1639B	insufficient preallocated memory for generation of precompiled header file (xxxx bytes required)

[Explanation]

Insufficient preallocated memory for generation of precompiled header file (xxxx bytes required).

W1640B	very large entity in program prevents generation of precompiled header file

[Explanation]

Very large entity in program prevents generation of precompiled header file.

W1645B	"xxxx" is an unrecognized __declspec attribute

[Explanation]

"xxxx" is an unrecognized __declspec attribute.

W1650B	calling convention specified here is ignored

[Explanation]

Calling convention specified here is ignored.

W1655B	the modifier "xxxx" is not allowed on this declaration

[Explanation]

The modifier "xxxx" is not allowed on this declaration.

W1656B	transfer of control into a try block

[Explanation]

Transfer of control into a try block.

W1657B	inline specification is incompatible with previous "entity" (declared at line xxxx)

[Explanation]

Inline specification is incompatible with previous "entity" (declared at line xxxx).

W1662B	call of pure virtual function

[Explanation]

Call of pure virtual function.

W1667B	"asm" function is nonstandard

[Explanation]

"asm" function is nonstandard.

W1668B	ellipsis with no explicit parameters is nonstandard

[Explanation]

Ellipsis with no explicit parameters is nonstandard.

W1669B	"&..." is nonstandard

[Explanation]

"&..." is nonstandard.

W1672B	temporary used for initial value of reference to const volatile (anachronism)

[Explanation]

Temporary used for initial value of reference to const volatile (anachronism).

W1674B	initial value of reference to const volatile must be an lvalue

[Explanation]

Initial value of reference to const volatile must be an lvalue.

W1676B	using out-of-scope declaration of entity-kind "entity" (declared at line xxxx)

[Explanation]

Using out-of-scope declaration of entity-kind "entity" (declared at line xxxx).

W1688B	"xxxx" not found on pack alignment stack

[Explanation]

"xxxx" not found on pack alignment stack.

W1689B	empty pack alignment stack

[Explanation]

Empty pack alignment stack.

W1691B	entity-kind "entity", required for copy that was eliminated, is inaccessible

[Explanation]

Entity-kind "entity", required for copy that was eliminated, is inaccessible.

W1692B	entity-kind "entity", required for copy that was eliminated, is not callable because reference parameter cannot be bound to rvalue

[Explanation]

Entity-kind "entity", required for copy that was eliminated, is not callable because reference parameter cannot be bound to rvalue.

W1708B	incrementing a bool value is deprecated

[Explanation]

Incrementing a bool value is deprecated.

W1711B	expression must have bool type (or be convertible to bool)

[Explanation]

Expression must have bool type (or be convertible to bool).

W1715B	__based does not precede a pointer operator, __based ignored.

[Explanation]

__based does not precede a pointer operator, __based ignored.

W1719B	mutable is not allowed

[Explanation]

Mutable is not allowed.

W1720B	redeclaration of entity-kind "entity" is not allowed to alter its access

[Explanation]

Redeclaration of entity-kind "entity" is not allowed to alter its access.

W1721B	nonstandard format string conversion

[Explanation]

Nonstandard format string conversion.

W1723B	use of alternative token "%:" appears to be unintended

[Explanation]

Use of alternative token "%:" appears to be unintended.

W1729B	invalid combination of DLL attributes

[Explanation]

Invalid combination of DLL attributes.

W1731B	array with incomplete element type is nonstandard

[Explanation]

Array with incomplete element type is nonstandard.

W1737B	using-declaration ignored -- it refers to the current namespace

[Explanation]

Using-declaration ignored -- it refers to the current namespace.

W1741B	using-declaration of entity-kind "entity" ignored

[Explanation]

Using-declaration of entity-kind "entity" ignored.

W1745B	memory attribute ignored

[Explanation]

Memory attribute ignored.

W1747B	memory attribute specified more than once

[Explanation]

Memory attribute specified more than once.

W1748B	calling convention specified more than once

[Explanation]

Calling convention specified more than once.

W1767B	conversion from pointer to smaller integer

[Explanation]

Conversion from pointer to smaller integer.

W1768B	exception specification for implicitly declared virtual entity-kind "entity" is incompatible with that of overridden entity-kind "entity"

[Explanation]

Exception specification for implicitly declared virtual entity-kind "entity" is incompatible with that of overridden entity-kind "entity".

W1772B	use of alternative token "<:" appears to be unintended

[Explanation]

Use of alternative token "<:" appears to be unintended.

W1780B	reference is to entity-kind "entity" (declared at line xxxx) -- under old for-init scoping rules it would have been entity-kind "entity" (declared at line xxxx)

[Explanation]

Reference is to entity-kind "entity" (declared at line xxxx) -- under old for-init scoping rules it would have been entity-kind "entity" (declared at line xxxx).

W1783B	empty comment interpreted as token-pasting operator "##"

[Explanation]

Empty comment interpreted as token-pasting operator "##".

W1802B	specifying a default argument when redeclaring an unreferenced function template is nonstandard

[Explanation]

Specifying a default argument when redeclaring an unreferenced function template is nonstandard.

W1806B	omission of exception specification is incompatible with entity-kind "entity" (declared at line xxxx)

[Explanation]

Omission of exception specification is incompatible with entity-kind "entity" (declared at line xxxx).

W1811B	const entity-kind "entity" requires an initializer -- class "type" has no explicitly declared default constructor

[Explanation]

Const entity-kind "entity" requires an initializer -- class "type" has no explicitly declared default constructor.

W1812B	const object requires an initializer -- class "type" has no explicitly declared default constructor

[Explanation]

Const object requires an initializer -- class "type" has no explicitly declared default constructor.

W1815B	type qualifier on return type is meaningless

[Explanation]

Type qualifier on return type is meaningless.

W1817B	static data member declaration is not allowed in this class

[Explanation]

Static data member declaration is not allowed in this class.

W1819B	"..." is not allowed

[Explanation]

"..." is not allowed.

W1825B	virtual inline entity-kind "entity" was never defined

[Explanation]

Virtual inline entity-kind "entity" was never defined.

W1826B	entity-kind "entity" was never referenced

[Explanation]

Entity-kind "entity" was never referenced.

W1829B	double used for "long double" in generated C code

[Explanation]

Double used for "long double" in generated C code.

W1830B	entity-kind "entity" has no corresponding operator deletexxxx (to be called if an exception is thrown during initialization of an allocated object)

[Explanation]

Entity-kind "entity" has no corresponding operator deletexxxx (to be called if an exception is thrown during initialization of an allocated object).

W1831B	support for placement delete is disabled

[Explanation]

Support for placement delete is disabled.

W1833B	pointer or reference to incomplete type is not allowed

[Explanation]

Pointer or reference to incomplete type is not allowed.

W1836B	returning reference to local variable

[Explanation]

Returning reference to local variable.

W1837B	omission of explicit type is nonstandard ("int" assumed)

[Explanation]

Omission of explicit type is nonstandard ("int" assumed).

When the -Jc option is not specified, this message is output to the following descriptions.

• A variable declaration without the type specifier.

• A function declaration (excluding "main") without the return type specifier.

When a variable or a function is declared, please explicitly specify the type or the return type.

The compiler assumes that "int" is specified when this warning is encountered.

W1848B	expression must have arithmetic or enum type

[Explanation]

Expression must have arithmetic or enum type.

W1850B	type of cast must be integral or enum

[Explanation]

Type of cast must be integral or enum.

W1860B	__declspec attributes ignored

[Explanation]

__declspec attributes ignored.

W1867B	declaration of "size_t" does not match the expected type "type"

[Explanation]

Declaration of "size_t" does not match the expected type "type".

W1870B	invalid multibyte character sequence

[Explanation]

Invalid multibyte character sequence.

W1902B	type qualifier ignored

[Explanation]

Type qualifier ignored.

W1912B	ambiguous class member reference -- entity-kind "entity" (declared at line xxxx) used in preference to entity-kind "entity" (declared at line xxxx)

[Explanation]

Ambiguous class member reference -- entity-kind "entity" (declared at line xxxx) used in preference to entity-kind "entity" (declared at line xxxx).

W1934B	a member with reference type is not allowed in a union

[Explanation]

A member with reference type is not allowed in a union.

W1936B	redeclaration of "entity" alters its access

[Explanation]

Redeclaration of "entity" alters its access.

W1938B	return type "int" omitted in declaration of function "main"

[Explanation]

Return type "int" omitted in declaration of function "main".

When the -Jc option is not specified, this message is output to the following descriptions.

• "main" function declaration without the return type specifier.

When "main" function is declared, please explicitly specify the return type.

The compiler assumes that the return type is "int" when this warning is encountered.

W1940B	missing return statement at end of non-void "entity"

[Explanation]

Missing return statement at end of non-void "entity".

W1941B	duplicate using-declaration of "entity" ignored

[Explanation]

Duplicate using-declaration of "entity" ignored.

W1942B	enum bit-fields are always unsigned, but enum type includes negative enumerator

[Explanation]

Enum bit-fields are always unsigned, but enum type includes negative enumerator.

W1946B	name following "template" must be a member template

[Explanation]

Name following "template" must be a member template.

W1947B	name following "template" must have a template argument list

[Explanation]

Name following "template" must have a template argument list.

W1948B	nonstandard local-class friend declaration. -- no prior declaration in the enclosing scope

[Explanation]

Nonstandard local-class friend declaration. -- no prior declaration in the enclosing scope.

W1949B	specifying a default argument when redeclaring an unreferenced function template is nonstandard

[Explanation]

Specifying a default argument when redeclaring an unreferenced function template is nonstandard.

W1951B	return type of function "main" must be "int"

[Explanation]

Return type of function "main" must be "int".

W1959B	declared size for bit-field is larger than the size of the bit-field-type.

[Explanation]

Declared size for bit-field is larger than the size of the bit-field-type.

W1961B	use of a type with no linkage to declare a variable with linkage

[Explanation]

Use of a type with no linkage to declare a variable with linkage.

W1962B	use of a type with no linkage to declare a function

[Explanation]

Use of a type with no linkage to declare a function.

W1966B	universal character name specifies an invalid character

[Explanation]

Universal character name specifies an invalid character.

W1967B	a universal character name cannot designate a character in the basic character set

[Explanation]

A universal character name cannot designate a character in the basic character set.

W1968B	this universal character is not allowed in an identifier

[Explanation]

This universal character is not allowed in an identifier.

W1970B	the qualifier on this friend declaration is ignored

[Explanation]

The qualifier on this friend declaration is ignored.

W1973B	"inline" used as a function qualifier is ignored

[Explanation]

"inline" used as a function qualifier is ignored.

W1984B	typedef name has already been declared (with similar type)

[Explanation]

typedef name has already been declared (with similar type).

W1985B	operator new and operator delete cannot be given internal linkage

[Explanation]

operator new and operator delete cannot be given internal linkage.

W1992B	extra braces are nonstandard

[Explanation]

extra braces are nonstandard.

W1994B	subtraction of pointer types %t1 and %t2 is nonstandard

[Explanation]

subtraction of pointer types %t1 and %t2 is nonstandard.

W1998B	%np2 is hidden by %no1 -- virtual function override intended?

[Explanation]

%np2 is hidden by %no1. --virtual function override intended.

W2001B	a storage class may not be specified here

[Explanation]

A storage class may not be specified here.

W2013B	a using-declaration may not name a constructor or destructor

[Explanation]

A using-declaration may not name a constructor or destructor.

W2028B	invalid redeclaration of nested class

[Explanation]

Invalid redeclaration of nested class.

W2030B	a variable with static storage duration cannot be defined within an inline function

[Explanation]

A variable with static storage duration cannot be defined within an inline function.

W2031B	an entity with internal linkage cannot be referenced within an inline function with external linkage

[Explanation]

An entity with internal linkage cannot be referenced within an inline function with external linkage.

W2039B	unrecognized STDC pragma

[Explanation]

Unrecognized STDC pragma.

W2040B	expected "ON", "OFF", or "DEFAULT"

[Explanation]

expected "ON", "OFF", or "DEFAULT".

W2041B	a STDC pragma may only appear between declarations in the global scope or before any statements or declarations in a block scope

[Explanation]

A STDC pragma may only appear between declarations in the global scope or before any statements or declarations in a block scope.

W2046B	floating-point value cannot be represented exactly

[Explanation]

Floating-point value cannot be represented exactly.

W2048B	conversion between real and imaginary yields zero

[Explanation]

Conversion between real and imaginary yields zero.

W2050B	imaginary *= imaginary sets the left-hand operand to zero

[Explanation]

"imaginary *= imaginary" sets the left-hand operand to zero.

W2051B	standard requires that "xxx" be given a type by a subsequent declaration ("int" assumed)

[Explanation]

Standard requires that "xxx" be given a type by a subsequent declaration ("int" assumed).

When the -Jc option is not specified, this message is output to the following descriptions.

- A parameter declaration without the type specifier.

When parameter is declared, please explicitly specify the type.

The compiler assumes that a parameter type is "int" when this warning is encountered.

W2052B	a definition is required for inline %n

[Explanation]

A definition is required for inline %n.

W2053B	conversion from integer to smaller pointer

[Explanation]

Conversion from integer to smaller pointer.

W2055B	types cannot be declared in anonymous unions

[Explanation]

Types cannot be declared in anonymous unions.

W2056B	returning pointer to local variable

[Explanation]

Returning pointer to local variable.

W2057B	returning pointer to local temporary

[Explanation]

Returning pointer to local temporary.

W2072B	a declaration cannot have a label

[Explanation]

A declaration cannot have a label.

W2073B	support for exported templates is disabled

[Explanation]

Support for exported templates is disabled.

W2093B	invalid attribute for parameter

[Explanation]

Invalid attribute for parameter.

W2097B	there is no attribute %sq

[Explanation]

There is no attribute %sq.

W2100B	the "packed" attribute is ignored in a typedef

[Explanation]

The "packed" attribute is ignored in a typedef.

W2102B	"goto *expr" is nonstandard

[Explanation]

"goto *expr" is nonstandard.

W2105B	#warning directive: %s

[Explanation]

#warning directive: %s.

W2108B	the "transparent_union" attribute is ignored on incomplete types

[Explanation]

The "transparent_union" attribute is ignored on incomplete types.

W2109B	%t cannot be transparent because %n does not have the same size as the union

[Explanation]

%t cannot be transparent because %n does not have the same size as the union.

W2110B	%t cannot be transparent because it has a field of type %t which is not the same size as the union

[Explanation]

%t cannot be transparent because it has a field of type %t which is not the same size as the union.

W2114B	declarations of local labels should only appear at the start of statement expressions

[Explanation]

Declarations of local labels should only appear at the start of statement expressions.

W2117B	an asm name is ignored in a typedef

[Explanation]

An asm name is ignored in a typedef.

W2119B	modifier letter '%s' ignored in asm operand

[Explanation]

Modifier letter '%s' ignored in asm operand.

W2139B	the "template" keyword used for syntactic disambiguation may only be used within a template

[Explanation]

The "template" keyword used for syntactic disambiguation may only be used within a template.

W2142B	attribute does not apply to non-function type %t

[Explanation]

Attribute does not apply to non-function type %t.

W2143B	arithmetic on pointer to void or function type

[Explanation]

Arithmetic on pointer to void or function type.

W2145B	%t1 would have been promoted to %t2 when passed through the ellipsis parameter; use the latter type instead

[Explanation]

%t1 would have been promoted to %t2 when passed through the ellipsis parameter; use the latter type instead.

W2152B	declaration aliased to unknown entity %sq

[Explanation]

Declaration aliased to unknown entity %sq.

W2153B	declaration does not match its alias %n

[Explanation]

Declaration does not match its alias %n.

W2155B	variable-length array field type will be treated as zero-length array field type

[Explanation]

Variable-length array field type will be treated as zero-length array field type.

W2156B	nonstandard cast on lvalue ignored

[Explanation]

Nonstandard cast on lvalue ignored.

W2159B	the auto specifier is ignored here (invalid in standard C/C++)

[Explanation]

The auto specifier is ignored here (invalid in standard C/C++).

W2160B	a reduction in alignment without the "packed" attribute is ignored

[Explanation]

A reduction in alignment without the "packed" attribute is ignored.

W2162B	excess initializers are ignored

[Explanation]

Excess initializers are ignored.

W2163B	va_start should only appear in a function with an ellipsis parameter

[Explanation]

Va_start should only appear in a function with an ellipsis parameter.

W2168B	an asm name is ignored on a non-register automatic variable

[Explanation]

An asm name is ignored on a non-register automatic variable.

W2169B	inline function also declared as an alias; definition ignored

[Explanation]

Inline function also declared as an alias; definition ignored.

W2170B	unrecognized UPC pragma

[Explanation]

Unrecognized UPC pragma.

W2178B	function returning shared is not allowed

[Explanation]

Function returning shared is not allowed.

W2185B	argument of upc_blocksizeof is a pointer to a shared type (not shared type itself)

[Explanation]

Argument of upc_blocksizeof is a pointer to a shared type (not shared type itself).

W2192B	null (zero) character in input line ignored

[Explanation]

Null (zero) character in input line ignored.

W2193B	null (zero) character in string or character constant

[Explanation]

Null (zero) character in string or character constant.

W2194B	null (zero) character in header name

[Explanation]

Null (zero) character in header name.

W2195B	declaration in for-initializer hides a declaration in the surrounding scope

[Explanation]

Declaration in for-initializer hides a declaration in the surrounding scope.

W2196B	the hidden declaration is %p

[Explanation]

The hidden declaration is %p.

W2197B	the prototype declaration of %nfd is ignored after this unprototyped redeclaration

[Explanation]

The prototype declaration of %nfd is ignored after this unprototyped redeclaration.

W2198B	attribute ignored on typedef of class or enum types

[Explanation]

Attribute ignored on typedef of class or enum types.

W2199B	variable declaration hides declaration in for-initializer

[Explanation]

Variable declaration hides declaration in for-initializer.

W2202B	call of zero constant ignored

[Explanation]

Call of zero constant ignored.

W2203B	parameter %sq may not be redeclared in a catch clause of function try block

[Explanation]

Parameter %sq may not be redeclared in a catch clause of function try block.

W2204B	the initial explicit specialization of %n must be declared in the namespace containing the template

[Explanation]

The initial explicit specialization of %n must be declared in the namespace containing the template.

W2205B	"cc" clobber ignored

[Explanation]

"cc" clobber ignored.

W2206B	"template" must be followed by an identifier

[Explanation]

"template" must be followed by an identifier.

W2210B	declaration of %sq hides handler parameter

[Explanation]

Declaration of %sq hides handler parameter.

W2211B	nonstandard cast to array type ignored

[Explanation]

Nonstandard cast to array type ignored.

W2213B	field uses tail padding of a base class

[Explanation]

Field uses tail padding of a base class.

W2214B	GNU C++ compilers may use bit field padding

[Explanation]

GNU C++ compilers may use bit field padding.

W2215B	use of "symbol" is deprecated

[Explanation]

Use of "symbol" is deprecated.

W2217B	unrecognized format function type "xxxx" ignored

[Explanation]

Unrecognized format function type "xxxx" ignored.

W2218B	base class "symbol" uses tail padding of base class "symbol"

[Explanation]

Base class "symbol" uses tail padding of base class "symbol".

W2220B	requested initialization priority is reserved for internal use

[Explanation]

Requested initialization priority is reserved for internal use.

W2221B	this anonymous union/struct field is hidden by "symbol"

[Explanation]

This anonymous union/struct field is hidden by "symbol".

W2222B	invalid error number

[Explanation]

Invalid error number.

W2223B	invalid error tag

[Explanation]

Invalid error tag.

W2224B	expected an error number or error tag

[Explanation]

Expected an error number or error tag.

W2225B	size of class is affected by tail padding

[Explanation]

Size of class is affected by tail padding.

W2235B	nonstandard conversion between pointer to function and pointer to data

[Explanation]

Nonstandard conversion between pointer to function and pointer to data.

W2257B	potentially narrowing conversion when compiled in an environment where int, long, or pointer types are 64 bits wide

[Explanation]

Potentially narrowing conversion when compiled in an environment where int, long, or pointer types are 64 bits wide.

W2258B	current value of pragma pack is xxxx

[Explanation]

Current value of pragma pack is xxxx.

W2259B	arguments for pragma pack(show) are ignored

[Explanation]

Arguments for pragma pack(show) are ignored.

W2262B	earlier __declspec(align(...)) ignored

[Explanation]

Earlier __declspec(align(...)) ignored.

W2272B	a throw expression may not have pointer-to-incomplete type

[Explanation]

A throw expression may not have pointer-to-incomplete type.

W2273B	alignment-of operator applied to incomplete type

[Explanation]

Alignment-of operator applied to incomplete type.

W2276B	unrecognized attribute "xxxx"

[Explanation]

Unrecognized attribute "xxxx".

W3007B	__interrupt is specified

[Explanation]

__interrupt is specified.

W3008B	__interrupt do not operate on function declarator

[Explanation]

__interrupt do not operate on function declarator.

W3009B	__interrupt operate class member function

[Explanation]

__interrupt operate class member function.

W3010B	__io is specified

[Explanation]

__io is specified.

W3011B	__io operate on function declarator

[Explanation]

__io operate on function declarator.

W3012B	__io operate member

[Explanation]

__io operate member.

W3016B	#pragma xxxx: too long identifier is specified

[Explanation]

#pragma xxxx: too long identifier is specified.

W3032B	#pragma ilm: already exist

[Explanation]

#pragma ilm: already exist.

W3038B	The specification 'mutable' cannot be used in EC++.

[Explanation]

The specification 'mutable' cannot be used in EC++.

W3039B	The function of the exceptional transaction cannot be used in EC++.

[Explanation]

The function of the exceptional transaction cannot be used in EC++.

W3040B	The namespace cannot be used in EC++.

[Explanation]

The namespace cannot be used in EC++.

W3041B	The template cannot be used in EC++.

[Explanation]

The template cannot be used in EC++.

W3042B	The multiple inheritance cannot be used in EC++.

[Explanation]

The multiple inheritance cannot be used in EC++.

W3043B	The virtual inheritance cannot be used in EC++.

[Explanation]

The virtual inheritance cannot be used in EC++.

W3044B	The operator 'const_cast' cannot be used in EC++.

[Explanation]

The operator 'const_cast' cannot be used in EC++.

W3045B	The operator 'dynamic_cast' cannot be used in EC++.

[Explanation]

The operator 'dynamic_cast' cannot be used in EC++.

W3046B	The operator 'static_cast' cannot be used in EC++.

[Explanation]

The operator 'static_cast' cannot be used in EC++.

W3047B	The operator 'reinterpret_cast' cannot be used in EC++.

[Explanation]

The operator 'reinterpret_cast' cannot be used in EC++.

W3048B	The operator 'typeid' cannot be used in EC++.

[Explanation]

The operator 'typeid' cannot be used in EC++.

W3054B	xxxx is not recognized a builtin function for the media instruction

[Explanation]

xxxx is not recognized a builtin function for the media instruction.

W3056B	value is immediately outside the range of the argument xxxx of xxxx (xxxx)

[Explanation]

Value is immediately outside the range of the argument xxxx of xxxx (xxxx).

W3057B	argument xxxx of xxxx should be an accumulator number defined by xxxx

[Explanation]

Argument xxxx of xxxx should be an accumulator number defined by xxxx.

W3058B	accumulator number of the argument xxxx of xxxx should be an even number

[Explanation]

Accumulator number of the argument xxxx of xxxx should be an even number.

W3059B	accumulator number of the argument xxxx of xxxx should be a multiple of four

[Explanation]

Accumulator number of the argument xxxx of xxxx should be a multiple of four.

E4001B	last line of file ends without a newline

[Explanation]

Last line of file ends without a newline.

E4002B	last line of file ends with a backslash

[Explanation]

Last line of file ends with a backslash.

E4006B	comment unclosed at end of file

[Explanation]

Comment unclosed at end of file.

E4007B	unrecognized token

[Explanation]

Unrecognized token.

E4008B	missing closing quote

[Explanation]

Missing closing quote.

E4010B	"#" not expected here

[Explanation]

"#" not expected here.

E4011B	unrecognized preprocessing directive

[Explanation]

Unrecognized preprocessing directive.

E4012B	parsing restarts here after previous syntax error

[Explanation]

Parsing restarts here after previous syntax error.

E4014B	extra text after expected end of preprocessing directive

[Explanation]

Extra text after expected end of preprocessing directive.

E4017B	expected a "]"

[Explanation]

Expected a "]".

E4018B	expected a ")"

[Explanation]

Expected a ")".

E4019B	extra text after expected end of number

[Explanation]

Extra text after expected end of number.

E4020B	identifier "xxxx" is undefined

[Explanation]

Identifier "xxxx" is undefined.

E4021B	type qualifiers are meaningless in this declaration

[Explanation]

Type qualifiers are meaningless in this declaration.

E4022B	invalid octal digit

[Explanation]

Invalid octal digit.

E4023B	integer constant is too large

[Explanation]

Integer constant is too large.

E4024B	Invalid octal digit

[Explanation]

Invalid octal digit.

E4025B	quoted string should contain at least one character

[Explanation]

Quoted string should contain at least one character.

E4026B	too many characters in character constant

[Explanation]

Too many characters in character constant.

E4027B	character value is out of range

[Explanation]

Character value is out of range.

E4028B	expression must have a constant value

[Explanation]

Expression must have a constant value.

E4029B	expected an expression

[Explanation]

Expected an expression.

E4030B	floating constant is out of range

[Explanation]

Floating constant is out of range.

E4031B	expression must have integral type

[Explanation]

Expression must have integral type.

E4032B	expression must have arithmetic type

[Explanation]

Expression must have arithmetic type.

E4033B	expected a line number

[Explanation]

Expected a line number.

E4034B	invalid line number

[Explanation]

Invalid line number.

E4036B	the #if for this directive is missing

[Explanation]

The #if for this directive is missing.

E4037B	the #endif for this directive is missing

[Explanation]

The #endif for this directive is missing.

E4038B	directive is not allowed -- an #else has already appeared

[Explanation]

Directive is not allowed -- an #else has already appeared.

E4039B	division by zero

[Explanation]

Division by zero.

E4040B	expected an identifier

[Explanation]

Expected an identifier.

E4041B	expression must have arithmetic or pointer type

[Explanation]

Expression must have arithmetic or pointer type.

E4042B	operand types are incompatible ("type" and "type")

[Explanation]

Operand types are incompatible ("type" and "type").

E4044B	expression must have pointer type

[Explanation]

Expression must have pointer type.

E4045B	#undef may not be used on this predefined name

[Explanation]

#undef may not be used on this predefined name.

E4046B	this predefined name may not be redefined

[Explanation]

This predefined name may not be redefined.

E4047B	incompatible redefinition of macro "entity" (declared at line xxxx)

[Explanation]

Incompatible redefinition of macro "entity" (declared at line xxxx).

E4049B	duplicate macro parameter name

[Explanation]

Duplicate macro parameter name.

E4050B	"##" may not be first in a macro definition

[Explanation]

"##" may not be first in a macro definition.

E4051B	"##" may not be last in a macro definition

[Explanation]

"##" may not be last in a macro definition.

E4052B	expected a macro parameter name

[Explanation]

Expected a macro parameter name.

E4053B	expected a ":"

[Explanation]

Expected a ":".

E4054B	too few arguments in macro invocation

[Explanation]

Too few arguments in macro invocation.

E4055B	too many arguments in macro invocation

[Explanation]

Too many arguments in macro invocation.

E4056B	operand of sizeof may not be a function

[Explanation]

Operand of sizeof may not be a function.

E4057B	this operator is not allowed in a constant expression

[Explanation]

This operator is not allowed in a constant expression.

E4058B	this operator is not allowed in a preprocessing expression

[Explanation]

This operator is not allowed in a preprocessing expression.

E4059B	function call is not allowed in a constant expression

[Explanation]

Function call is not allowed in a constant expression.

E4060B	this operator is not allowed in an integral constant expression

[Explanation]

This operator is not allowed in an integral constant expression.

E4061B	integer operation result is out of range

[Explanation]

Integer operation result is out of range.

E4062B	shift count is negative

[Explanation]

Shift count is negative.

E4063B	shift count is too large

[Explanation]

Shift count is too large.

E4064B	declaration does not declare anything

[Explanation]

Declaration does not declare anything.

E4065B	expected a ";"

[Explanation]

Expected a ";".

E4066B	enumeration value is out of "int" range

[Explanation]

Enumeration value is out of "int" range.

E4067B	expected a "}"

[Explanation]

Expected a "}".

E4069B	integer conversion resulted in truncation

[Explanation]

Integer conversion resulted in truncation.

E4070B	incomplete type is not allowed

[Explanation]

Incomplete type is not allowed.

E4071B	operand of sizeof may not be a bit field

[Explanation]

Operand of sizeof may not be a bit field.

E4075B	operand of "*" must be a pointer

[Explanation]

Operand of "*" must be a pointer.

E4077B	this declaration has no storage class or type specifier

[Explanation]

This declaration has no storage class or type specifier.

E4078B	a parameter declaration may not have an initializer

[Explanation]

A parameter declaration may not have an initializer.

E4079B	expected a type specifier

[Explanation]

Expected a type specifier.

E4080B	a storage class may not be specified here

[Explanation]

A storage class may not be specified here.

E4081B	more than one storage class may not be specified

[Explanation]

More than one storage class may not be specified.

E4082B	storage class is not first

[Explanation]

Storage class is not first.

E4083B	type qualifier specified more than once

[Explanation]

Type qualifier specified more than once.

E4084B	invalid combination of type specifiers

[Explanation]

Invalid combination of type specifiers.

E4085B	invalid storage class for a parameter

[Explanation]

Invalid storage class for a parameter.

E4086B	invalid storage class for a function

[Explanation]

Invalid storage class for a function.

E4087B	a type specifier may not be used here

[Explanation]

A type specifier may not be used here.

E4088B	array of functions is not allowed

[Explanation]

Array of functions is not allowed.

E4089B	array of void is not allowed

[Explanation]

Array of void is not allowed.

E4090B	function returning function is not allowed

[Explanation]

Function returning function is not allowed.

E4091B	function returning array is not allowed

[Explanation]

Function returning array is not allowed.

E4092B	identifier-list parameters may only be used in a function definition

[Explanation]

Identifier-list parameters may only be used in a function definition.

E4093B	function type may not come from a typedef

[Explanation]

Function type may not come from a typedef.

E4094B	the size of an array must be greater than zero

[Explanation]

The size of an array must be greater than zero.

E4095B	array is too large

[Explanation]

Array is too large.

E4096B	a translation unit must contain at least one declaration

[Explanation]

A translation unit must contain at least one declaration.

E4097B	a function may not return a value of this type

[Explanation]

A function may not return a value of this type.

E4098B	an array may not have elements of this type

[Explanation]

An array may not have elements of this type.

E4099B	a declaration here must declare a parameter

[Explanation]

A declaration here must declare a parameter.

E4100B	duplicate parameter name

[Explanation]

Duplicate parameter name.

E4101B	"xxxx" has already been declared in the current scope

[Explanation]

"xxxx" has already been declared in the current scope.

E4102B	forward declaration of enum type is nonstandard

[Explanation]

Forward declaration of enum type is nonstandard.

E4103B	class is too large

[Explanation]

Class is too large.

E4104B	struct or union is too large

[Explanation]

Struct or union is too large.

E4105B	invalid size for bit field

[Explanation]

Invalid size for bit field.

E4106B	invalid type for a bit field

[Explanation]

Invalid type for a bit field.

E4107B	zero-length bit field must be unnamed

[Explanation]

Zero-length bit field must be unnamed.

E4109B	expression must have (pointer-to-) function type

[Explanation]

Expression must have (pointer-to-) function type.

E4110B	expected either a definition or a tag name

[Explanation]

Expected either a definition or a tag name.

E4112B	expected "while"

[Explanation]

Expected "while".

E4114B	entity-kind "entity" was referenced but not defined

[Explanation]

Entity-kind "entity" was referenced but not defined.

E4115B	a continue statement may only be used within a loop

[Explanation]

A continue statement may only be used within a loop.

E4116B	a break statement may only be used within a loop or switch

[Explanation]

A break statement may only be used within a loop or switch.

E4117B	non-void function "entity" should return a value

[Explanation]

Non-void function "entity" should return a value.

E4118B	a void function may not return a value

[Explanation]

A void function may not return a value.

E4119B	cast to type "type" is not allowed

[Explanation]

Cast to type "type" is not allowed.

E4120B	return value type does not match the function type

[Explanation]

Return value type does not match the function type.

E4121B	a case label may only be used within a switch

[Explanation]

A case label may only be used within a switch.

E4122B	a default label may only be used within a switch

[Explanation]

A default label may only be used within a switch.

E4123B	case label value has already appeared in this switch

[Explanation]

Case label value has already appeared in this switch.

E4124B	default label has already appeared in this switch

[Explanation]

default label has already appeared in this switch.

E4125B	expected a "("

[Explanation]

Expected a "(".

E4126B	expression must be an lvalue

[Explanation]

Expression must be an lvalue.

E4127B	expected a statement

[Explanation]

Expected a statement.

E4129B	a block-scope function may only have extern storage class

[Explanation]

A block-scope function may only have extern storage class.

E4130B	expected a "{"

[Explanation]

Expected a "{".

E4131B	expression must have pointer-to-class type

[Explanation]

Expression must have pointer-to-class type.

E4132B	expression must have pointer-to-struct-or-union type

[Explanation]

Expression must have pointer-to-struct-or-union type.

E4133B	expected a member name

[Explanation]

Expected a member name.

E4134B	expected a field name

[Explanation]

Expected a field name.

E4135B	entity-kind "entity" has no member "xxxx"

[Explanation]

Entity-kind "entity" has no member "xxxx".

E4136B	entity-kind "entity" has no field "xxxx"

[Explanation]

Entity-kind "entity" has no field "xxxx".

E4137B	expression must be a modifiable lvalue

[Explanation]

Expression must be a modifiable lvalue.

E4138B	taking the address of a register variable is not allowed

[Explanation]

Taking the address of a register variable is not allowed.

E4139B	taking the address of a bit field is not allowed

[Explanation]

Taking the address of a bit field is not allowed.

E4140B	too many arguments in function call

[Explanation]

Too many arguments in function call.

E4141B	unnamed prototyped parameters not allowed when body is present

[Explanation]

Unnamed prototyped parameters not allowed when body is present.

E4142B	expression must have pointer-to-object type

[Explanation]

Expression must have pointer-to-object type.

E4144B	a value of type "type" cannot be used to initialize an entity of type "type"

[Explanation]

A value of type "type" cannot be used to initialize an entity of type "type".

E4145B	entity-kind "entity" may not be initialized

[Explanation]

Entity-kind "entity" may not be initialized.

E4146B	too many initializer values

[Explanation]

Too many initializer values.

E4147B	declaration is incompatible with entity-kind "entity" (declared at line xxxx)

[Explanation]

Declaration is incompatible with entity-kind "entity" (declared at line xxxx).

E4148B	entity-kind "entity" has already been initialized

[Explanation]

Entity-kind "entity" has already been initialized.

E4149B	a global-scope declaration may not have this storage class

[Explanation]

A global-scope declaration may not have this storage class.

E4150B	a type name may not be redeclared as a parameter

[Explanation]

A type name may not be redeclared as a parameter.

E4151B	a typedef name may not be redeclared as a parameter

[Explanation]

A typedef name may not be redeclared as a parameter.

E4153B	expression must have class type

[Explanation]

Expression must have class type.

E4154B	expression must have struct or union type

[Explanation]

Expression must have struct or union type.

E4157B	expression must be an integral constant expression

[Explanation]

Expression must be an integral constant expression.

E4158B	expression must be an lvalue or a function designator

[Explanation]

Expression must be an lvalue or a function designator.

E4159B	declaration is incompatible with previous "entity" (declared at line xxxx)

[Explanation]

Declaration is incompatible with previous "entity" (declared at line xxxx).

E4160B	name conflicts with previously used external name "xxxx"

[Explanation]

Name conflicts with previously used external name "xxxx".

E4161B	unrecognized #pragma

[Explanation]

Unrecognized #pragma.

E4165B	too few arguments in function call

[Explanation]

Too few arguments in function call.

E4166B	invalid floating constant

[Explanation]

Invalid floating constant.

E4167B	argument of type "type" is incompatible with parameter of type "type"

[Explanation]

Argument of type "type" is incompatible with parameter of type "type".

E4168B	a function type is not allowed here

[Explanation]

A function type is not allowed here.

E4169B	expected a declaration

[Explanation]

Expected a declaration.

E4171B	invalid type conversion

[Explanation]

Invalid type conversion.

E4172B	external/internal linkage conflict with previous declaration

[Explanation]

External/internal linkage conflict with previous declaration.

E4173B	floating-point value does not fit in required integral type

[Explanation]

Floating-point value does not fit in required integral type.

E4179B	right operand of "%" is zero

[Explanation]

Right operand of "%" is zero.

E4183B	type of cast must be integral

[Explanation]

Type of cast must be integral.

E4184B	type of cast must be arithmetic or pointer

[Explanation]

Type of cast must be arithmetic or pointer.

E4194B	expected an asm string

[Explanation]

Expected an asm string.

E4195B	an asm function must be prototyped

[Explanation]

An asm function must be prototyped.

E4196B	an asm function may not have an ellipsis

[Explanation]

An asm function may not have an ellipsis.

E4220B	integral value does not fit in required floating-point type

[Explanation]

Integral value does not fit in required floating-point type.

E4221B	floating-point value does not fit in required floating-point type

[Explanation]

Floating-point value does not fit in required floating-point type.

E4222B	floating-point operation result is out of range

[Explanation]

Floating-point operation result is out of range.

E4227B	macro recursion

[Explanation]

Macro recursion.

E4228B	trailing comma is nonstandard

[Explanation]

Trailing comma is nonstandard.

E4230B	nonstandard type for a bit field

[Explanation]

Nonstandard type for a bit field.

E4235B	variable "xxxx" was declared with a never-completed type

[Explanation]

Variable "xxxx" was declared with a never-completed type.

E4238B	invalid specifier on a parameter

[Explanation]

Invalid specifier on a parameter.

E4239B	invalid specifier outside a class declaration

[Explanation]

Invalid specifier outside a class declaration.

E4240B	duplicate specifier in declaration

[Explanation]

Duplicate specifier in declaration.

E4241B	a union is not allowed to have a base class

[Explanation]

A union is not allowed to have a base class.

E4242B	multiple access control specifiers are not allowed

[Explanation]

Multiple access control specifiers are not allowed.

E4243B	class or struct definition is missing

[Explanation]

Class or struct definition is missing.

E4244B	qualified name is not a member of class "type" or its base classes

[Explanation]

Qualified name is not a member of class "type" or its base classes.

E4245B	a nonstatic member reference must be relative to a specific object

[Explanation]

A nonstatic member reference must be relative to a specific object.

E4246B	a nonstatic data member may not be defined outside its class

[Explanation]

A nonstatic data member may not be defined outside its class.

E4247B	entity-kind "entity" has already been defined

[Explanation]

Entity-kind "entity" has already been defined.

E4248B	pointer to reference is not allowed

[Explanation]

Pointer to reference is not allowed.

E4249B	reference to reference is not allowed

[Explanation]

Reference to reference is not allowed.

E4250B	reference to void is not allowed

[Explanation]

Reference to void is not allowed.

E4251B	array of reference is not allowed

[Explanation]

Array of reference is not allowed.

E4252B	reference entity-kind "entity" requires an initializer

[Explanation]

Reference entity-kind "entity" requires an initializer.

E4253B	expected a ","

[Explanation]

Expected a ",".

E4254B	type name is not allowed

[Explanation]

Type name is not allowed.

E4255B	type definition is not allowed

[Explanation]

Type definition is not allowed.

E4256B	invalid redeclaration of type name "entity" (declared at line xxxx)

[Explanation]

Invalid redeclaration of type name "entity" (declared at line xxxx).

E4257B	const entity-kind "entity" requires an initializer

[Explanation]

Const entity-kind "entity" requires an initializer.

E4258B	"this" may only be used inside a nonstatic member function

[Explanation]

"this" may only be used inside a nonstatic member function.

E4259B	constant value is not known

[Explanation]

Constant value is not known.

E4262B	not a class or struct name

[Explanation]

Not a class or struct name.

E4263B	duplicate base class name

[Explanation]

Duplicate base class name.

E4264B	invalid base class

[Explanation]

Invalid base class.

E4265B	entity-kind "entity" is inaccessible

[Explanation]

Entity-kind "entity" is inaccessible.

E4266B	"entity" is ambiguous

[Explanation]

"entity" is ambiguous.

E4267B	old-style parameter list (anachronism)

[Explanation]

Old-style parameter list (anachronism).

E4268B	declaration may not appear after executable statement in block

[Explanation]

Declaration may not appear after executable statement in block.

E4269B	conversion to inaccessible base class "type" is not allowed

[Explanation]

Conversion to inaccessible base class "type" is not allowed.

E4274B	improperly terminated macro invocation

[Explanation]

Improperly terminated macro invocation.

E4276B	name followed by "::" must be a class or namespace name

[Explanation]

Name followed by "::" must be a class or namespace name.

E4277B	invalid friend declaration

[Explanation]

Invalid friend declaration.

E4278B	a constructor or destructor may not return a value

[Explanation]

A constructor or destructor may not return a value.

E4279B	invalid destructor declaration

[Explanation]

Invalid destructor declaration.

E4280B	declaration of a member with the same name as its class

[Explanation]

Declaration of a member with the same name as its class.

E4281B	global-scope qualifier (leading "::") is not allowed

[Explanation]

Global-scope qualifier (leading "::") is not allowed.

E4282B	the global scope has no "xxxx"

[Explanation]

The global scope has no "xxxx".

E4283B	qualified name is not allowed

[Explanation]

Qualified name is not allowed.

E4284B	NULL reference is not allowed

[Explanation]

NULL reference is not allowed.

E4285B	initialization with "{...}" is not allowed for object of type "type"

[Explanation]

Initialization with "{...}" is not allowed for object of type "type".

E4286B	base class "type" is ambiguous

[Explanation]

Base class "type" is ambiguous.

E4287B	derived class "type" contains more than one instance of class "type"

[Explanation]

Derived class "type" contains more than one instance of class "type".

E4288B	cannot convert pointer to base class "type" to pointer to derived class "type" -- base class is virtual

[Explanation]

Cannot convert pointer to base class "type" to pointer to derived class "type" -- base class is virtual.

E4289B	no instance of constructor "entity" matches the argument list

[Explanation]

No instance of constructor "entity" matches the argument list.

E4290B	copy constructor for class "type" is ambiguous

[Explanation]

Copy constructor for class "type" is ambiguous.

E4291B	no default constructor exists for class "type"

[Explanation]

No default constructor exists for class "type".

E4292B	"xxxx" is not a nonstatic data member or base class of class "type"

[Explanation]

"xxxx" is not a nonstatic data member or base class of class "type".

E4293B	indirect nonvirtual base class is not allowed

[Explanation]

Indirect nonvirtual base class is not allowed.

E4294B	invalid union member -- class "type" has a disallowed member function

[Explanation]

Invalid union member -- class "type" has a disallowed member function.

E4296B	invalid use of non-lvalue array

[Explanation]

Invalid use of non-lvalue array.

E4297B	expected an operator

[Explanation]

Expected an operator.

E4298B	inherited member is not allowed

[Explanation]

Inherited member is not allowed.

E4299B	cannot determine which instance of entity-kind "entity" is intended

[Explanation]

Cannot determine which instance of entity-kind "entity" is intended.

E4300B	a pointer to a bound function may only be used to call the function

[Explanation]

A pointer to a bound function may only be used to call the function.

E4301B	typedef name has already been declared (with same type)

[Explanation]

Typedef name has already been declared (with same type).

E4302B	entity-kind "entity" has already been defined

[Explanation]

Entity-kind "entity" has already been defined.

E4304B	no instance of entity-kind "entity" matches the argument list

[Explanation]

No instance of entity-kind "entity" matches the argument list.

E4305B	type definition is not allowed in function return type declaration

[Explanation]

Type definition is not allowed in function return type declaration.

E4306B	default argument not at end of parameter list

[Explanation]

Default argument not at end of parameter list.

E4307B	redefinition of default argument

[Explanation]

Redefinition of default argument.

E4308B	more than one instance of entity-kind "entity" matches the argument list

[Explanation]

More than one instance of entity-kind "entity" matches the argument list.

E4309B	more than one instance of constructor "entity" matches the argument list

[Explanation]

More than one instance of constructor "entity" matches the argument list.

E4310B	default argument of type "type" is incompatible with parameter of type "type"

[Explanation]

Default argument of type "type" is incompatible with parameter of type "type".

E4311B	cannot overload functions distinguished by return type alone

[Explanation]

Cannot overload functions distinguished by return type alone.

E4312B	no suitable user-defined conversion from "type" to "type" exists

[Explanation]

No suitable user-defined conversion from "type" to "type" exists.

E4313B	type qualifier is not allowed on this function

[Explanation]

Type qualifier is not allowed on this function.

E4314B	only nonstatic member functions may be virtual

[Explanation]

Only nonstatic member functions may be virtual.

E4315B	the object has type qualifiers that are not compatible with the member function

[Explanation]

The object has type qualifiers that are not compatible with the member function.

E4316B	program too large to compile (too many virtual functions)

[Explanation]

Program too large to compile (too many virtual functions).

E4317B	return type is not identical to nor covariant with return type "type" of overridden virtual function entity-kind "entity"

[Explanation]

Return type is not identical to nor covariant with return type "type" of overridden virtual function entity-kind "entity".

E4318B	override of virtual entity-kind ""entity"" is ambiguous

[Explanation]

Override of virtual entity-kind "entity" is ambiguous.

E4319B	pure specifier ("= 0") allowed only on virtual functions

[Explanation]

Pure specifier ("= 0") allowed only on virtual functions.

E4320B	badly-formed pure specifier (only "= 0" is allowed)

[Explanation]

Badly-formed pure specifier (only "= 0" is allowed).

E4321B	data member initializer is not allowed

[Explanation]

Data member initializer is not allowed.

E4322B	object of abstract class type "type" is not allowed

[Explanation]

Object of abstract class type "type" is not allowed.

E4323B	function returning abstract class "type" is not allowed

[Explanation]

Function returning abstract class "type" is not allowed.

E4325B	inline specifier allowed on function declarations only

[Explanation]

Inline specifier allowed on function declarations only.

E4326B	"inline" is not allowed

[Explanation]

"inline" is not allowed.

E4327B	invalid storage class for an inline function

[Explanation]

Invalid storage class for an inline function.

E4328B	invalid storage class for a class member

[Explanation]

Invalid storage class for a class member.

E4329B	local class member entity-kind "entity" requires a definition

[Explanation]

Local class member entity-kind "entity" requires a definition.

E4330B	entity-kind "entity" is inaccessible

[Explanation]

Entity-kind "entity" is inaccessible.

E4332B	class "type" has no copy constructor to copy a const object

[Explanation]

Class "type" has no copy constructor to copy a const object.

E4333B	defining an implicitly declared member function is not allowed

[Explanation]

Defining an implicitly declared member function is not allowed.

E4334B	class "type" has no suitable copy constructor

[Explanation]

Class "type" has no suitable copy constructor.

E4335B	linkage specification is not allowed

[Explanation]

Linkage specification is not allowed.

E4336B	unknown external linkage specification

[Explanation]

Unknown external linkage specification.

E4337B	linkage specification is incompatible with previous "entity" (declared at line xxxx)

[Explanation]

Linkage specification is incompatible with previous "entity" (declared at line xxxx).

E4338B	more than one instance of overloaded function "entity" has "C" linkage

[Explanation]

More than one instance of overloaded function "entity" has "C" linkage.

E4339B	class "type" has more than one default constructor

[Explanation]

Class "type" has more than one default constructor.

E4341B	"operator xxxx" must be a member function

[Explanation]

"operator xxxx" must be a member function.

E4342B	operator may not be a static member function

[Explanation]

Operator may not be a static member function.

E4343B	no arguments allowed on user-defined conversion

[Explanation]

No arguments allowed on user-defined conversion.

E4344B	too many parameters for this operator function

[Explanation]

Too many parameters for this operator function.

E4345B	too few parameters for this operator function

[Explanation]

Too few parameters for this operator function.

E4346B	nonmember operator requires a parameter with class type

[Explanation]

Nonmember operator requires a parameter with class type.

E4347B	default argument is not allowed

[Explanation]

Default argument is not allowed.

E4348B	more than one user-defined conversion from "type" to "type" applies

[Explanation]

More than one user-defined conversion from "type" to "type" applies.

E4349B	no operator "xxxx" matches these operands

[Explanation]

No operator "xxxx" matches these operands.

E4350B	more than one operator "xxxx" matches these operands

[Explanation]

More than one operator "xxxx" matches these operands.

E4351B	first parameter of allocation function must be of type "size_t"

[Explanation]

First parameter of allocation function must be of type "size_t".

E4352B	allocation function requires "void *" return type

[Explanation]

Allocation function requires "void *" return type.

E4353B	deallocation function requires "void" return type

[Explanation]

Deallocation function requires "void" return type.

E4354B	first parameter of deallocation function must be of type void *

[Explanation]

First parameter of deallocation function must be of type void *.

E4356B	type must be an object type

[Explanation]

Type must be an object type.

E4357B	base class "type" has already been initialized

[Explanation]

Base class "type" has already been initialized.

E4358B	base class name required -- "type" assumed (anachronism)

[Explanation]

Base class name required -- "type" assumed (anachronism).

E4359B	entity-kind "entity" has already been initialized

[Explanation]

Entity-kind "entity" has already been initialized.

E4360B	name of member or base class is missing

[Explanation]

Name of member or base class is missing.

E4361B	assignment to "this" (anachronism)

[Explanation]

Assignment to "this" (anachronism).

E4362B	"overload" keyword used (anachronism)

[Explanation]

"overload" keyword used (anachronism).

E4363B	invalid anonymous union -- nonpublic member is not allowed

[Explanation]

Invalid anonymous union -- nonpublic member is not allowed.

E4364B	invalid anonymous union -- member function is not allowed

[Explanation]

Invalid anonymous union -- member function is not allowed.

E4365B	anonymous union at global or namespace scope must be declared static

[Explanation]

Anonymous union at global or namespace scope must be declared static.

E4366B	entity-kind "entity" provides no initializer for

[Explanation]

Entity-kind "entity" provides no initializer for.

E4367B	implicitly generated constructor for class "type" cannot initialize

[Explanation]

Implicitly generated constructor for class "type" cannot initialize.

E4369B	entity-kind "entity" has an uninitialized const or reference member

[Explanation]

Entity-kind "entity" has an uninitialized const or reference member.

E4371B	class "type" has no assignment operator to copy a const object

[Explanation]

Class "type" has no assignment operator to copy a const object.

E4372B	class "type" has no suitable assignment operator

[Explanation]

Class "type" has no suitable assignment operator.

E4373B	ambiguous assignment operator for class "type"

[Explanation]

Ambiguous assignment operator for class "type".

E4375B	declaration requires a typedef name

[Explanation]

Declaration requires a typedef name.

E4377B	"virtual" is not allowed

[Explanation]

"virtual" is not allowed.

E4378B	"static" is not allowed

[Explanation]

"static" is not allowed.

E4379B	cast of bound function to normal function pointer (anachronism)

[Explanation]

Cast of bound function to normal function pointer (anachronism).

E4380B	expression must have pointer-to-member type

[Explanation]

Expression must have pointer-to-member type.

E4381B	extra ";" ignored

[Explanation]

Extra ";" ignored.

E4382B	nonstandard member constant declaration (standard form is a static const integral member)

[Explanation]

Nonstandard member constant declaration (standard form is a static const integral member).

E4384B	no instance of overloaded "entity" matches the argument list

[Explanation]

No instance of overloaded "entity" matches the argument list.

E4386B	no instance of entity-kind "entity" matches the required type

[Explanation]

No instance of entity-kind "entity" matches the required type.

E4387B	delete array size expression used (anachronism)

[Explanation]

Delete array size expression used (anachronism).

E4388B	"operator->" for class "type" returns invalid type "type"

[Explanation]

"operator->" for class "type" returns invalid type "type".

E4389B	a cast to abstract class "type" is not allowed

[Explanation]

A cast to abstract class "type" is not allowed.

E4390B	function "main" may not be called or have its address taken

[Explanation]

Function "main" may not be called or have its address taken.

E4391B	a new-initializer may not be specified for an array

[Explanation]

A new-initializer may not be specified for an array.

E4392B	member function "entity" may not be redeclared outside its class

[Explanation]

Member function "entity" may not be redeclared outside its class.

E4393B	pointer to incomplete class type is not allowed

[Explanation]

Pointer to incomplete class type is not allowed.

E4394B	reference to local variable of enclosing function is not allowed

[Explanation]

Reference to local variable of enclosing function is not allowed.

E4395B	single-argument function used for postfix "xxxx" (anachronism)

[Explanation]

Single-argument function used for postfix "xxxx" (anachronism).

E4397B	implicitly generated assignment operator cannot copy

[Explanation]

Implicitly generated assignment operator cannot copy.

E4403B	entity-kind "entity" has already been declared

[Explanation]

Entity-kind "entity" has already been declared.

E4404B	function "main" may not be declared inline

[Explanation]

Function "main" may not be declared inline.

E4405B	member function with the same name as its class must be a constructor

[Explanation]

Member function with the same name as its class must be a constructor.

E4406B	using nested entity-kind "entity" (anachronism)

[Explanation]

Using nested entity-kind "entity" (anachronism).

E4407B	a destructor may not have parameters

[Explanation]

A destructor may not have parameters.

E4408B	copy constructor for class "type" may not have a parameter of type "type"

[Explanation]

Copy constructor for class "type" may not have a parameter of type "type".

E4409B	entity-kind "entity" returns incomplete type "type"

[Explanation]

Entity-kind "entity" returns incomplete type "type".

E4410B	protected entity-kind "entity" is not accessible through a "type" pointer or object

[Explanation]

Protected entity-kind "entity" is not accessible through a "type" pointer or object.

E4411B	a parameter is not allowed

[Explanation]

A parameter is not allowed.

E4412B	an "asm" declaration is not allowed here

[Explanation]

An "asm" declaration is not allowed here.

E4413B	no suitable conversion function from "type" to "type" exists

[Explanation]

No suitable conversion function from "type" to "type" exists.

E4415B	no suitable constructor exists to convert from "type" to "type"

[Explanation]

No suitable constructor exists to convert from "type" to "type".

E4416B	more than one constructor applies to convert from "type" to "type"

[Explanation]

More than one constructor applies to convert from "type" to "type".

E4417B	more than one conversion function from "type" to "type" applies

[Explanation]

More than one conversion function from "type" to "type" applies.

E4418B	more than one conversion function from "type" to a built-in type applies

[Explanation]

More than one conversion function from "type" to a built-in type applies.

E4424B	a constructor or destructor may not have its address taken

[Explanation]

A constructor or destructor may not have its address taken.

E4425B	dollar sign ("$") used in identifier

[Explanation]

Dollar sign ("$") used in identifier.

E4426B	temporary used for initial value of reference to non-const (anachronism)

[Explanation]

Temporary used for initial value of reference to non-const (anachronism).

E4427B	qualified name is not allowed in member declaration

[Explanation]

Qualified name is not allowed in member declaration.

E4428B	enumerated type mixed with another type (anachronism)

[Explanation]

Enumerated type mixed with another type (anachronism).

E4429B	the size of an array in "new" must be non-negative

[Explanation]

The size of an array in "new" must be non-negative.

E4432B	"enum" declaration is not allowed

[Explanation]

"enum" declaration is not allowed.

E4433B	qualifiers dropped in binding reference of type "type" to initializer of type "type"

[Explanation]

Qualifiers dropped in binding reference of type "type" to initializer of type "type".

E4434B	a reference of type "type" (not const-qualified) cannot be initialized with a value of type "type"

[Explanation]

A reference of type "type" (not const-qualified) cannot be initialized with a value of type "type".

E4435B	a pointer to function may not be deleted

[Explanation]

A pointer to function may not be deleted.

E4436B	conversion function must be a nonstatic member function

[Explanation]

Conversion function must be a nonstatic member function.

E4437B	template declaration is not allowed here

[Explanation]

Template declaration is not allowed here.

E4438B	expected a "<"

[Explanation]

Expected a "<".

E4439B	expected a ">"

[Explanation]

Expected a ">".

E4440B	template parameter declaration is missing

[Explanation]

Template parameter declaration is missing.

E4441B	argument list for entity-kind "entity" is missing

[Explanation]

Argument list for entity-kind "entity" is missing.

E4442B	too few arguments for entity-kind "entity"

[Explanation]

Too few arguments for entity-kind "entity".

E4443B	too many arguments for entity-kind "entity"

[Explanation]

Too many arguments for entity-kind "entity".

E4445B	entity-kind "entity" is not used in declaring the parameter types of entity-kind "entity"

[Explanation]

Entity-kind "entity" is not used in declaring the parameter types of entity-kind "entity".

E4446B	two nested types have the same name: "entity" and "entity" (declared at line xxxx) (cfront compatibility)

[Explanation]

Two nested types have the same name: "entity" and "entity" (declared at line xxxx) (cfront compatibility).

E4447B	global "entity" was declared after nested "entity" (declared at line xxxx) (cfront compatibility)

[Explanation]

Global "entity" was declared after nested "entity" (declared at line xxxx) (cfront compatibility).

E4449B	more than one instance of entity-kind "entity" matches the required type

[Explanation]

More than one instance of entity-kind "entity" matches the required type.

E4450B	the type "long long" is nonstandard

[Explanation]

The type "long long" is nonstandard.

E4451B	omission of "xxxx" is nonstandard

[Explanation]

Omission of "xxxx" is nonstandard.

E4452B	return type may not be specified on a conversion function

[Explanation]

Return type may not be specified on a conversion function.

E4456B	excessive recursion at instantiation of entity-kind "entity"

[Explanation]

Excessive recursion at instantiation of entity-kind "entity".

E4457B	"xxxx" is not a function or static data member

[Explanation]

"xxxx" is not a function or static data member.

E4458B	argument of type "type" is incompatible with template parameter of type "type"

[Explanation]

Argument of type "type" is incompatible with template parameter of type "type".

E4459B	initialization requiring a temporary or conversion is not allowed

[Explanation]

Initialization requiring a temporary or conversion is not allowed.

E4461B	initial value of reference to non-const must be an lvalue

[Explanation]

Initial value of reference to non-const must be an lvalue.

E4463B	"template" is not allowed

[Explanation]

"template" is not allowed.

E4464B	"type" is not a class template

[Explanation]

"type" is not a class template.

E4466B	"main" is not a valid name for a function template

[Explanation]

"main" is not a valid name for a function template.

E4467B	invalid reference to entity-kind "entity" (union/nonunion mismatch)

[Explanation]

Invalid reference to entity-kind "entity" (union/nonunion mismatch).

E4468B	a template argument may not reference a local type

[Explanation]

A template argument may not reference a local type.

E4469B	tag kind of xxxx is incompatible with declaration of entity-kind "entity" (declared at line xxxx)

[Explanation]

Tag kind of xxxx is incompatible with declaration of entity-kind "entity" (declared at line xxxx).

E4470B	the global scope has no tag named "xxxx"

[Explanation]

The global scope has no tag named "xxxx".

E4471B	entity-kind "entity" has no tag member named "xxxx"

[Explanation]

Entity-kind "entity" has no tag member named "xxxx".

E4473B	entity-kind "entity" may be used only in pointer-to-member declaration

[Explanation]

Entity-kind "entity" may be used only in pointer-to-member declaration.

E4475B	a template argument may not reference a non-external entity

[Explanation]

A template argument may not reference a non-external entity.

E4476B	name followed by "::~" must be a class name or a type name

[Explanation]

Name followed by "::~" must be a class name or a type name.

E4477B	destructor name does not match name of class "type"

[Explanation]

Destructor name does not match name of class "type".

E4478B	type used as destructor name does not match type "type"

[Explanation]

Type used as destructor name does not match type "type".

E4481B	invalid storage class for a template declaration

[Explanation]

Invalid storage class for a template declaration.

E4484B	invalid explicit instantiation declaration

[Explanation]

Invalid explicit instantiation declaration.

E4485B	entity-kind "entity" is not an entity that can be instantiated

[Explanation]

Entity-kind "entity" is not an entity that can be instantiated.

E4486B	compiler generated entity-kind "entity" cannot be explicitly instantiated

[Explanation]

Compiler generated entity-kind "entity" cannot be explicitly instantiated.

E4487B	inline entity-kind "entity" cannot be explicitly instantiated

[Explanation]

Inline entity-kind "entity" cannot be explicitly instantiated.

E4488B	pure virtual entity-kind "entity" cannot be explicitly instantiated

[Explanation]

Pure virtual entity-kind "entity" cannot be explicitly instantiated.

E4489B	entity-kind "entity" cannot be instantiated -- no template definition was supplied

[Explanation]

Entity-kind "entity" cannot be instantiated -- no template definition was supplied.

E4490B	entity-kind "entity" cannot be instantiated -- it has been explicitly specialized

[Explanation]

Entity-kind "entity" cannot be instantiated -- it has been explicitly specialized.

E4493B	no instance of entity-kind "entity" matches the specified type

[Explanation]

No instance of entity-kind "entity" matches the specified type.

E4496B	template parameter "xxxx" may not be redeclared in this scope

[Explanation]

Template parameter "xxxx" may not be redeclared in this scope.

E4498B	template argument list must match the parameter list

[Explanation]

Template argument list must match the parameter list.

E4500B	extra parameter of postfix "operator xxxx" must be of type "int"

[Explanation]

Extra parameter of postfix "operator xxxx" must be of type "int".

E4501B	an operator name must be declared as a function

[Explanation]

An operator name must be declared as a function.

E4502B	operator name is not allowed

[Explanation]

Operator name is not allowed.

E4503B	entity-kind "entity" cannot be specialized in the current scope

[Explanation]

Entity-kind "entity" cannot be specialized in the current scope.

E4504B	nonstandard form for taking the address of a member function

[Explanation]

Nonstandard form for taking the address of a member function.

E4505B	too few template parameters -- does not match previous declaration

[Explanation]

Too few template parameters -- does not match previous declaration.

E4506B	too many template parameters -- does not match previous declaration

[Explanation]

Too many template parameters -- does not match previous declaration.

E4507B	function template for operator delete(void *) is not allowed

[Explanation]

Function template for operator delete(void *) is not allowed.

E4508B	class template and template parameter may not have the same name

[Explanation]

Class template and template parameter may not have the same name.

E4510B	a template argument may not reference an unnamed type

[Explanation]

A template argument may not reference an unnamed type.

E4511B	enumerated type is not allowed

[Explanation]

Enumerated type is not allowed.

E4512B	type qualifier on a reference type is not allowed

[Explanation]

Type qualifier on a reference type is not allowed.

E4513B	a value of type "type" cannot be assigned to an entity of type "type"

[Explanation]

A value of type "type" cannot be assigned to an entity of type "type".

E4515B	cannot convert to incomplete class "type"

[Explanation]

Cannot convert to incomplete class "type".

E4516B	const object requires an initializer

[Explanation]

Const object requires an initializer.

E4517B	object has an uninitialized const or reference member

[Explanation]

Object has an uninitialized const or reference member.

E4518B	nonstandard preprocessing directive

[Explanation]

Nonstandard preprocessing directive.

E4519B	entity-kind "entity" may not have a template argument list

[Explanation]

Entity-kind "entity" may not have a template argument list.

E4520B	initialization with "{...}" expected for aggregate object

[Explanation]

Initialization with "{...}" expected for aggregate object.

E4521B	pointer-to-member selection class types are incompatible ("type" and "type")

[Explanation]

Pointer-to-member selection class types are incompatible ("type" and "type").

E4522B	pointless friend declaration

[Explanation]

Pointless friend declaration.

E4525B	a dependent statement may not be a declaration

[Explanation]

A dependent statement may not be a declaration.

E4526B	a parameter may not have void type

[Explanation]

A parameter may not have void type.

E4529B	this operator is not allowed in a template argument expression

[Explanation]

This operator is not allowed in a template argument expression.

E4530B	try block requires at least one handler

[Explanation]

Try block requires at least one handler.

E4531B	handler requires an exception declaration

[Explanation]

Handler requires an exception declaration.

E4532B	handler is masked by default handler

[Explanation]

Handler is masked by default handler.

E4536B	exception specification is incompatible with that of previous entity-kind "entity" (declared at line xxxx)xxxx

[Explanation]

Exception specification is incompatible with that of previous entity-kind "entity" (declared at line xxxx)xxxx.

E4540B	support for exception handling is disabled

[Explanation]

Support for exception handling is disabled.

E4541B	omission of exception specification is incompatible with previous entity-kind "entity" (declared at line xxxx)

[Explanation]

Omission of exception specification is incompatible with previous entity-kind "entity" (declared at line xxxx).

E4543B	non-arithmetic operation not allowed in nontype template argument

[Explanation]

Non-arithmetic operation not allowed in nontype template argument.

E4544B	use of a local type to declare a nonlocal variable

[Explanation]

Use of a local type to declare a nonlocal variable.

E4545B	use of a local type to declare a function

[Explanation]

Use of a local type to declare a function.

E4546B	transfer of control bypasses initialization of

[Explanation]

Transfer of control bypasses initialization of.

E4548B	transfer of control into an exception handler

[Explanation]

Transfer of control into an exception handler.

E4551B	entity-kind "entity" cannot be defined in the current scope

[Explanation]

Entity-kind "entity" cannot be defined in the current scope.

E4552B	exception specification is not allowed

[Explanation]

Exception specification is not allowed.

E4555B	tag kind of xxxx is incompatible with template parameter of type "type"

[Explanation]

Tag kind of xxxx is incompatible with template parameter of type "type".

E4556B	function template for operator new(size_t) is not allowed

[Explanation]

Function template for operator new(size_t) is not allowed.

E4558B	pointer to member of type "type" is not allowed

[Explanation]

Pointer to member of type "type" is not allowed.

E4559B	ellipsis is not allowed in operator function parameter list

[Explanation]

Ellipsis is not allowed in operator function parameter list.

E4560B	"entity" is reserved for future use as a keyword

[Explanation]

"entity" is reserved for future use as a keyword.

E4598B	a template parameter may not have void type

[Explanation]

A template parameter may not have void type.

E4599B	excessive recursive instantiation of entity-kind "entity" due to instantiate-all mode

[Explanation]

Excessive recursive instantiation of entity-kind "entity" due to instantiate-all mode.

E4601B	a throw expression may not have void type

[Explanation]

A throw expression may not have void type.

E4603B	parameter of abstract class type "type" is not allowed

[Explanation]

Parameter of abstract class type "type" is not allowed.

E4604B	array of abstract class "type" is not allowed

[Explanation]

Array of abstract class "type" is not allowed.

E4605B	floating-point template parameter is nonstandard

[Explanation]

Floating-point template parameter is nonstandard.

E4606B	this pragma must immediately precede a declaration

[Explanation]

This pragma must immediately precede a declaration.

E4607B	this pragma must immediately precede a statement

[Explanation]

This pragma must immediately precede a statement.

E4608B	this pragma must immediately precede a declaration or statement

[Explanation]

This pragma must immediately precede a declaration or statement.

E4609B	this kind of pragma may not be used here

[Explanation]

This kind of pragma may not be used here.

E4612B	specific definition of inline template function must precede its first use

[Explanation]

Specific definition of inline template function must precede its first use.

E4615B	parameter type involves pointer to array of unknown bound

[Explanation]

Parameter type involves pointer to array of unknown bound.

E4616B	parameter type involves reference to array of unknown bound

[Explanation]

Parameter type involves reference to array of unknown bound.

E4619B	nonstandard unnamed field

[Explanation]

Nonstandard unnamed field.

E4620B	nonstandard unnamed member

[Explanation]

Nonstandard unnamed member.

E4624B	"xxxx" is not a type name

[Explanation]

"xxxx" is not a type name.

E4643B	"restrict" is not allowed

[Explanation]

"restrict" is not allowed.

E4644B	a pointer or reference to function type may not be qualified by "restrict"

[Explanation]

A pointer or reference to function type may not be qualified by "restrict".

E4646B	a calling convention modifier may not be specified here

[Explanation]

A calling convention modifier may not be specified here.

E4647B	conflicting calling convention modifiers

[Explanation]

Conflicting calling convention modifiers.

E4651B	a calling convention may not be followed by a nested declarator

[Explanation]

A calling convention may not be followed by a nested declarator.

E4654B	declaration modifiers are incompatible with previous declaration

[Explanation]

Declaration modifiers are incompatible with previous declaration.

E4655B	the modifier "xxxx" is not allowed on this declaration

[Explanation]

The modifier "xxxx" is not allowed on this declaration.

E4656B	transfer of control into a try block

[Explanation]

Transfer of control into a try block.

E4658B	closing brace of template definition not found

[Explanation]

Closing brace of template definition not found.

E4660B	invalid packing alignment value

[Explanation]

Invalid packing alignment value.

E4661B	expected an integer constant

[Explanation]

Expected an integer constant.

E4663B	invalid source file identifier string

[Explanation]

Invalid source file identifier string.

E4664B	a class template cannot be defined in a friend declaration

[Explanation]

A class template cannot be defined in a friend declaration.

E4665B	"asm" is not allowed

[Explanation]

"asm" is not allowed.

E4666B	"asm" must be used with a function definition

[Explanation]

"asm" must be used with a function definition.

E4667B	"asm" function is nonstandard

[Explanation]

"asm" function is nonstandard.

E4668B	ellipsis with no explicit parameters is nonstandard

[Explanation]

Ellipsis with no explicit parameters is nonstandard.

E4669B	"&..." is nonstandard

[Explanation]

"&..." is nonstandard.

E4670B	invalid use of "&..."

[Explanation]

Invalid use of "&...".

E4672B	temporary used for initial value of reference to const volatile (anachronism)

[Explanation]

Temporary used for initial value of reference to const volatile (anachronism).

E4673B	a reference of type "type" cannot be initialized with a value of type "type"

[Explanation]

A reference of type "type" cannot be initialized with a value of type "type".

E4674B	initial value of reference to const volatile must be an lvalue

[Explanation]

Initial value of reference to const volatile must be an lvalue.

E4681B	expected __except or __finally

[Explanation]

Expected __except or __finally.

E4682B	a __leave statement may only be used within a __try

[Explanation]

A __leave statement may only be used within a __try.

E4688B	"xxxx" not found on pack alignment stack

[Explanation]

"xxxx" not found on pack alignment stack.

E4689B	empty pack alignment stack

[Explanation]

Empty pack alignment stack.

E4691B	entity-kind "entity", required for copy that was eliminated, is inaccessible

[Explanation]

Entity-kind "entity", required for copy that was eliminated, is inaccessible.

E4692B	entity-kind "entity", required for copy that was eliminated, is not callable because reference parameter cannot be bound to rvalue

[Explanation]

Entity-kind "entity", required for copy that was eliminated, is not callable because reference parameter cannot be bound to rvalue.

E4693B	<typeinfo> must be included before typeid is used

[Explanation]

<typeinfo> must be included before typeid is used.

E4694B	xxxx cannot cast away const or other type qualifiers

[Explanation]

xxxx cannot cast away const or other type qualifiers.

E4695B	the type in a dynamic_cast must be a pointer or reference to a complete class type, or void *

[Explanation]

The type in a dynamic_cast must be a pointer or reference to a complete class type, or void *.

E4696B	the operand of a pointer dynamic_cast must be a pointer to a complete class type

[Explanation]

The operand of a pointer dynamic_cast must be a pointer to a complete class type.

E4697B	the operand of a reference dynamic_cast must be an lvalue of a complete class type

[Explanation]

The operand of a reference dynamic_cast must be an lvalue of a complete class type.

E4698B	the operand of a runtime dynamic_cast must have a polymorphic class type

[Explanation]

The operand of a runtime dynamic_cast must have a polymorphic class type.

E4701B	an array type is not allowed here

[Explanation]

An array type is not allowed here.

E4702B	expected an "="

[Explanation]

Expected an "=".

E4703B	expected a declarator in condition declaration

[Explanation]

Expected a declarator in condition declaration.

E4704B	"xxxx", declared in condition, may not be redeclared in this scope

[Explanation]

"xxxx", declared in condition, may not be redeclared in this scope.

E4705B	default template arguments are not allowed for function templates

[Explanation]

Default template arguments are not allowed for function templates.

E4706B	expected a "," or ">"

[Explanation]

Expected a "," or ">".

E4707B	expected a template parameter list

[Explanation]

Expected a template parameter list.

E4709B	bool type is not allowed

[Explanation]

Bool type is not allowed.

E4710B	offset of base class "entity" within class "entity" is too large

[Explanation]

Offset of base class "entity" within class "entity" is too large.

E4711B	expression must have bool type (or be convertible to bool)

[Explanation]

Expression must have bool type (or be convertible to bool).

E4713B	entity-kind "entity" is not a variable name

[Explanation]

Entity-kind "entity" is not a variable name

E4714B	__based modifier is not allowed here

[Explanation]

__based modifier is not allowed here.

E4716B	variable in __based modifier must have pointer type

[Explanation]

Variable in __based modifier must have pointer type.

E4717B	the type in a const_cast must be a pointer, reference, or pointer to member to an object type

[Explanation]

The type in a const_cast must be a pointer, reference, or pointer to member to an object type.

E4718B	a const_cast can only adjust type qualifiers: it cannot change the underlying type

[Explanation]

A const_cast can only adjust type qualifiers: it cannot change the underlying type.

E4719B	mutable is not allowed

[Explanation]

Mutable is not allowed.

E4720B	redeclaration of entity-kind "entity" is not allowed to alter its access

[Explanation]

Redeclaration of entity-kind "entity" is not allowed to alter its access.

E4724B	namespace definition is not allowed

[Explanation]

Namespace definition is not allowed.

E4725B	name must be a namespace name

[Explanation]

Name must be a namespace name.

E4726B	namespace alias definition is not allowed

[Explanation]

Namespace alias definition is not allowed.

E4727B	namespace-qualified name is required

[Explanation]

Namespace-qualified name is required.

E4728B	a namespace name is not allowed

[Explanation]

A namespace name is not allowed.

E4730B	entity-kind "entity" is not a class template

[Explanation]

Entity-kind "entity" is not a class template.

E4731B	array with incomplete element type is nonstandard

[Explanation]

Array with incomplete element type is nonstandard.

E4732B	allocation operator may not be declared in a namespace

[Explanation]

Allocation operator may not be declared in a namespace.

E4733B	deallocation operator may not be declared in a namespace

[Explanation]

Deallocation operator may not be declared in a namespace.

E4734B	entity-kind "entity" conflicts with using-declaration of entity-kind "entity"

[Explanation]

Entity-kind "entity" conflicts with using-declaration of entity-kind "entity".

E4735B	using-declaration of entity-kind "entity" conflicts with entity-kind "entity" (declared at line xxxx)

[Explanation]

Using-declaration of entity-kind "entity" conflicts with entity-kind "entity" (declared at line xxxx).

E4738B	a class-qualified name is required

[Explanation]

A class-qualified name is required.

E4742B	entity-kind "entity" has no actual member "xxxx"

[Explanation]

Entity-kind "entity" has no actual member "xxxx".

E4744B	incompatible memory attributes specified

[Explanation]

Incompatible memory attributes specified.

E4746B	memory attribute may not be followed by a nested declarator

[Explanation]

Memory attribute may not be followed by a nested declarator.

E4749B	a type qualifier is not allowed

[Explanation]

A type qualifier is not allowed.

E4750B	entity-kind "entity" (declared at line xxxx) was used before its template was declared

[Explanation]

Entity-kind "entity" (declared at line xxxx) was used before its template was declared.

E4751B	static and nonstatic member functions with same parameter types cannot be overloaded

[Explanation]

Static and nonstatic member functions with same parameter types cannot be overloaded.

E4752B	no prior declaration of entity-kind "entity"

[Explanation]

No prior declaration of entity-kind "entity".

E4753B	a template-id is not allowed

[Explanation]

A template-id is not allowed.

E4754B	a class-qualified name is not allowed

[Explanation]

A class-qualified name is not allowed.

E4755B	entity-kind "entity" may not be redeclared in the current scope

[Explanation]

Entity-kind "entity" may not be redeclared in the current scope.

E4756B	qualified name is not allowed in namespace member declaration

[Explanation]

Qualified name is not allowed in namespace member declaration.

E4757B	entity-kind "entity" is not a type name

[Explanation]

Entity-kind "entity" is not a type name.

E4758B	explicit instantiation is not allowed in the current scope

[Explanation]

Explicit instantiation is not allowed in the current scope.

E4759B	entity-kind "entity" cannot be explicitly instantiated in the current scope

[Explanation]

Entity-kind "entity" cannot be explicitly instantiated in the current scope.

E4760B	entity-kind "entity" explicitly instantiated more than once

[Explanation]

Entity-kind "entity" explicitly instantiated more than once.

E4761B	typename may only be used within a template

[Explanation]

Typename may only be used within a template.

E4765B	nonstandard character at start of object-like macro definition

[Explanation]

Nonstandard character at start of object-like macro definition.

E4766B	exception specification for virtual entity-kind "entity" is incompatible with that of overridden entity-kind "entity"

[Explanation]

Exception specification for virtual entity-kind "entity" is incompatible with that of overridden entity-kind "entity".

E4769B	"entity", implicitly called from entity-kind "entity", is ambiguous

[Explanation]

"entity", implicitly called from entity-kind "entity", is ambiguous.

E4771B	"explicit" is not allowed

[Explanation]

"explicit" is not allowed.

E4772B	declaration conflicts with "xxxx" (reserved class name)

[Explanation]

Declaration conflicts with "xxxx" (reserved class name).

E4773B	only "()" is allowed as initializer for array entity-kind "entity"

[Explanation]

Only "()" is allowed as initializer for array entity-kind "entity".

E4774B	"virtual" is not allowed in a function template declaration

[Explanation]

"virtual" is not allowed in a function template declaration.

E4775B	invalid anonymous union -- class member template is not allowed

[Explanation]

Invalid anonymous union -- class member template is not allowed.

E4776B	template nesting depth does not match the previous declaration of entity-kind "entity"

[Explanation]

Template nesting depth does not match the previous declaration of entity-kind "entity".

E4777B	this declaration cannot have multiple "template <...>" clauses

[Explanation]

This declaration cannot have multiple "template <...>" clauses.

E4779B	"xxxx", declared in for-loop initialization, may not be redeclared in this scope

[Explanation]

"xxxx", declared in for-loop initialization, may not be redeclared in this scope.

E4782B	definition of virtual entity-kind "entity" is required here

[Explanation]

Definition of virtual entity-kind "entity" is required here.

E4784B	a storage class is not allowed in a friend declaration

[Explanation]

A storage class is not allowed in a friend declaration.

E4785B	template parameter list for "entity" is not allowed in this declaration

[Explanation]

Template parameter list for "entity" is not allowed in this declaration.

E4786B	entity-kind "entity" is not a valid member class or function template

[Explanation]

Entity-kind "entity" is not a valid member class or function template.

E4787B	not a valid member class or function template declaration

[Explanation]

Not a valid member class or function template declaration.

E4788B	a template declaration containing a template parameter list may not be followed by an explicit specialization declaration

[Explanation]

A template declaration containing a template parameter list may not be followed by an explicit specialization declaration.

E4789B	explicit specialization of entity-kind "entity" must precede the first use of entity-kind "entity"

[Explanation]

Explicit specialization of entity-kind "entity" must precede the first use of entity-kind "entity".

E4790B	explicit specialization is not allowed in the current scope

[Explanation]

Explicit specialization is not allowed in the current scope.

E4791B	partial specialization of entity-kind "entity" is not allowed

[Explanation]

Partial specialization of entity-kind "entity" is not allowed.

E4792B	entity-kind "entity" is not an entity that can be explicitly specialized

[Explanation]

Entity-kind "entity" is not an entity that can be explicitly specialized.

E4793B	explicit specialization of entity-kind "entity" must precede its first use

[Explanation]

Explicit specialization of entity-kind "entity" must precede its first use.

E4794B	template parameter xxxx may not be used in an elaborated type specifier

[Explanation]

Template parameter xxxx may not be used in an elaborated type specifier.

E4795B	specializing entity-kind "entity" requires "template<>" syntax

[Explanation]

Specializing entity-kind "entity" requires "template<>" syntax.

E4799B	specializing entity-kind "entity" without "template<>" syntax is nonstandard

[Explanation]

Specializing entity-kind "entity" without "template<>" syntax is nonstandard.

E4800B	this declaration may not have extern "C" linkage

[Explanation]

This declaration may not have extern "C" linkage.

E4801B	"xxxx" is not a class or function template name in the current scope

[Explanation]

"xxxx" is not a class or function template name in the current scope.

E4802B	specifying a default argument when redeclaring an unreferenced function template is nonstandard

[Explanation]

Specifying a default argument when redeclaring an unreferenced function template is nonstandard.

E4803B	specifying a default argument when redeclaring an already referenced function template is not allowed

[Explanation]

Specifying a default argument when redeclaring an already referenced function template is not allowed.

E4804B	cannot convert pointer to member of base class "type" to pointer to member of derived class "type" -- base class is virtual

[Explanation]

Cannot convert pointer to member of base class "type" to pointer to member of derived class "type" -- base class is virtual.

E4805B	exception specification is incompatible with that of entity-kind "entity" (declared at line xxxx)xxxx

[Explanation]

Exception specification is incompatible with that of entity-kind "entity" (declared at line xxxx)xxxx.

E4806B	omission of exception specification is incompatible with entity-kind "entity" (declared at line xxxx)

[Explanation]

Omission of exception specification is incompatible with entity-kind "entity" (declared at line xxxx).

E4807B	the parse of this expression has changed between the point at which it appeared in the program and the point at which the expression was evaluated -- "typename" may be required to resolve the ambiguity

[Explanation]

The parse of this expression has changed between the point at which it appeared in the program and the point at which the expression was evaluated -- "typename" may be required to resolve the ambiguity.

E4808B	default-initialization of reference is not allowed

[Explanation]

Default-initialization of reference is not allowed.

E4809B	uninitialized entity-kind "entity" has a const member

[Explanation]

Uninitialized entity-kind "entity" has a const member.

E4810B	uninitialized base class "type" has a const member

[Explanation]

Uninitialized base class "type" has a const member.

E4811B	const entity-kind "entity" requires an initializer -- class "type" has no explicitly declared default constructor

[Explanation]

Const entity-kind "entity" requires an initializer -- class "type" has no explicitly declared default constructor.

E4812B	const object requires an initializer -- class "type" has no explicitly declared default constructor

[Explanation]

Const object requires an initializer -- class "type" has no explicitly declared default constructor.

E4816B	in a function definition a type qualifier on a "void" return type is not allowed

[Explanation]

In a function definition a type qualifier on a "void" return type is not allowed.

E4817B	static data member declaration is not allowed in this class

[Explanation]

Static data member declaration is not allowed in this class.

E4818B	template instantiation resulted in an invalid function declaration

[Explanation]

Template instantiation resulted in an invalid function declaration.

E4819B	"..." is not allowed

[Explanation]

"..." is not allowed.

E4821B	extern inline entity-kind "entity" was referenced but not defined

[Explanation]

Extern inline entity-kind "entity" was referenced but not defined.

E4822B	invalid destructor name for type "type"

[Explanation]

Invalid destructor name for type "type".

E4824B	destructor reference is ambiguous -- both entity-kind "entity" and entity-kind "entity" could be used

[Explanation]

Destructor reference is ambiguous -- both entity-kind "entity" and entity-kind "entity" could be used.

E4827B	only one member of a union may be specified in a constructor initializer list

[Explanation]

Only one member of a union may be specified in a constructor initializer list.

E4828B	support for "new[]" and "delete[]" is disabled

[Explanation]

Support for "new[]" and "delete[]" is disabled.

E4832B	no appropriate operator delete is visible

[Explanation]

No appropriate operator delete is visible.

E4833B	pointer or reference to incomplete type is not allowed

[Explanation]

Pointer or reference to incomplete type is not allowed.

E4834B	invalid partial specialization -- entity-kind "entity" is already fully specialized

[Explanation]

Invalid partial specialization -- entity-kind "entity" is already fully specialized.

E4835B	incompatible exception specifications

[Explanation]

Incompatible exception specifications.

E4837B	omission of explicit type is nonstandard ("int" assumed)

[Explanation]

Omission of explicit type is nonstandard ("int" assumed).

E4838B	more than one partial specialization matches the template argument list of entity-kind "entity"

[Explanation]

More than one partial specialization matches the template argument list of entity-kind "entity".

E4840B	a template argument list is not allowed in a declaration of a primary template

[Explanation]

A template argument list is not allowed in a declaration of a primary template.

E4841B	partial specializations may not have default template arguments

[Explanation]

Partial specializations may not have default template arguments.

E4842B	entity-kind "entity" is not used in template argument list of entity-kind "entity"

[Explanation]

Entity-kind "entity" is not used in template argument list of entity-kind "entity".

E4843B	the type of partial specialization template parameter entity-kind "entity" depends on another template parameter

[Explanation]

The type of partial specialization template parameter entity-kind "entity" depends on another template parameter.

E4844B	the template argument list of the partial specialization includes a nontype argument whose type depends on a template parameter

[Explanation]

The template argument list of the partial specialization includes a nontype argument whose type depends on a template parameter.

E4845B	this partial specialization would have been used to instantiate entity-kind "entity"

[Explanation]

This partial specialization would have been used to instantiate entity-kind "entity".

E4846B	this partial specialization would have been made the instantiation of entity-kind "entity" ambiguous

[Explanation]

This partial specialization would have been made the instantiation of entity-kind "entity" ambiguous.

E4847B	expression must have integral or enum type

[Explanation]

Expression must have integral or enum type.

E4848B	expression must have arithmetic or enum type

[Explanation]

Expression must have arithmetic or enum type.

E4849B	expression must have arithmetic, enum, or pointer type

[Explanation]

Expression must have arithmetic, enum, or pointer type.

E4850B	type of cast must be integral or enum

[Explanation]

Type of cast must be integral or enum.

E4851B	type of cast must be arithmetic, enum, or pointer

[Explanation]

Type of cast must be arithmetic, enum, or pointer.

E4852B	expression must be a pointer to a complete object type

[Explanation]

Expression must be a pointer to a complete object type.

E4853B	a partial specialization of a member class template must be declared in the class of which it is a member

[Explanation]

A partial specialization of a member class template must be declared in the class of which it is a member.

E4854B	a partial specialization nontype argument must be the name of a nontype parameter or a constant

[Explanation]

A partial specialization nontype argument must be the name of a nontype parameter or a constant.

E4855B	return type is not identical to return type "type" of overridden virtual function entity-kind "entity"

[Explanation]

Return type is not identical to return type "type" of overridden virtual function entity-kind "entity".

E4857B	a partial specialization of a class template must be declared in the namespace of which it is a member

[Explanation]

A partial specialization of a class template must be declared in the namespace of which it is a member.

E4861B	invalid character in input line

[Explanation]

Invalid character in input line.

E4862B	function returns incomplete type "type"

[Explanation]

Function returns incomplete type "type".

E4864B	xxxx is not a template

[Explanation]

xxxx is not a template.

E4865B	a friend declaration may not declare a partial specialization

[Explanation]

A friend declaration may not declare a partial specialization.

E4868B	space required between adjacent ">" delimiters of nested template argument lists (">>" is the right shift operator)

[Explanation]

Space required between adjacent ">" delimiters of nested template argument lists (">>" is the right shift operator).

E4871B	template instantiation resulted in unexpected function type of "type" (the meaning of a name may have changed since the template declaration -- the type of the template is "type")

[Explanation]

Template instantiation resulted in unexpected function type of "type" (the meaning of a name may have changed since the template declaration -- the type of the template is "type").

E4872B	ambiguous guiding declaration -- more than one function template "entity" matches type "type"

[Explanation]

Ambiguous guiding declaration -- more than one function template "entity" matches type "type".

E4873B	non-integral operation not allowed in nontype template argument

[Explanation]

Non-integral operation not allowed in nontype template argument.

E4875B	Embedded C++ does not support templates

[Explanation]

Embedded C++ does not support templates.

E4876B	Embedded C++ does not support exception handling

[Explanation]

Embedded C++ does not support exception handling.

E4877B	Embedded C++ does not support namespaces

[Explanation]

Embedded C++ does not support namespaces.

E4878B	Embedded C++ does not support run-time type information

[Explanation]

Embedded C++ does not support run-time type information.

E4879B	Embedded C++ does not support the new cast syntax

[Explanation]

Embedded C++ does not support the new cast syntax.

E4880B	Embedded C++ does not support using-declarations

[Explanation]

Embedded C++ does not support using-declarations.

E4881B	Embedded C++ does not support "mutable"

[Explanation]

Embedded C++ does not support "mutable".

E4882B	Embedded C++ does not support multiple or virtual inheritance

[Explanation]

Embedded C++ does not support multiple or virtual inheritance.

E4884B	pointer-to-member representation "xxxx" has already been set for entity-kind "entity"

[Explanation]

Pointer-to-member representation "xxxx" has already been set for entity-kind "entity"

E4885B	"type" cannot be used to designate constructor for "type"

[Explanation]

"type" cannot be used to designate constructor for "type".

E4886B	invalid suffix on integral constant

[Explanation]

Invalid suffix on integral constant.

E4887B	operand of __uuidof must have a class type for which __declspec(uuid("...")) has been specified

[Explanation]

Operand of __uuidof must have a class type for which __declspec(uuid("...")) has been specified.

E4888B	invalid GUID string in __declspec(uuid("..."))

[Explanation]

Invalid GUID string in __declspec(uuid("...")).

E4890B	variable length array with unspecified bound is not allowed

[Explanation]

Variable length array with unspecified bound is not allowed.

E4891B	an explicit template argument list is not allowed on this declaration

[Explanation]

An explicit template argument list is not allowed on this declaration.

E4892B	an entity with linkage cannot have a variably modified type

[Explanation]

An entity with linkage cannot have a variably modified type.

E4893B	a variable length array cannot have static storage duration

[Explanation]

A variable length array cannot have static storage duration.

E4894B	entity-kind "entity" is not a template

[Explanation]

Entity-kind "entity" is not a template.

E4896B	expected a template argument

[Explanation]

Expected a template argument.

E4898B	nonmember operator requires a parameter with class or enum type

[Explanation]

Nonmember operator requires a parameter with class or enum type.

E4900B	using-declaration of entity-kind "entity" is not allowed

[Explanation]

Using-declaration of entity-kind "entity" is not allowed.

E4901B	qualifier of destructor name "type" does not match type "type"

[Explanation]

Qualifier of destructor name "type" does not match type "type".

E4905B	incorrect property specification; correct form is __declspec(property(get=name1, put=name2))

[Explanation]

Incorrect property specification; correct form is __declspec(property(get=name1, put=name2)).

E4906B	property has already been specified

[Explanation]

Property has already been specified.

E4907B	__declspec(property) is not allowed on this declaration

[Explanation]

__declspec(property) is not allowed on this declaration.

E4908B	member is declared with __declspec(property), but no "get" function was specified

[Explanation]

Member is declared with __declspec(property), but no "get" function was specified.

E4909B	the __declspec(property) "get" function "xxxx" is missing

[Explanation]

The __declspec(property) "get" function "xxxx" is missing.

E4910B	member is declared with __declspec(property), but no "put" function was specified

[Explanation]

Member is declared with __declspec(property), but no "put" function was specified.

E4911B	the __declspec(property) "put" function "xxxx" is missing

[Explanation]

The __declspec(property) "put" function "xxxx" is missing.

E4912B	ambiguous class member reference -- entity-kind "entity" (declared at line xxxx) used in preference to entity-kind "entity" (declared at line xxxx)

[Explanation]

Ambiguous class member reference -- entity-kind "entity" (declared at line xxxx) used in preference to entity-kind "entity" (declared at line xxxx).

E4913B	missing or invalid segment name in __declspec(allocate("..."))

[Explanation]

Missing or invalid segment name in __declspec(allocate("...")).

E4914B	__declspec(allocate) is not allowed on this declaration

[Explanation]

__declspec(allocate) is not allowed on this declaration.

E4915B	a segment name has already been specified

[Explanation]

A segment name has already been specified.

E4916B	cannot convert pointer to member of derived class "type" to pointer to member of base class "type" -- base class is virtual

[Explanation]

Cannot convert pointer to member of derived class "type" to pointer to member of base class "type" -- base class is virtual.

E4925B	a type qualifier cannot be applied to a function type

[Explanation]

A type qualifier cannot be applied to a function type.

E4928B	incorrect use of va_start

[Explanation]

Incorrect use of va_start.

E4929B	incorrect use of va_arg

[Explanation]

Incorrect use of va_arg.

E4930B	incorrect use of va_end

[Explanation]

Incorrect use of va_end.

E4934B	a member with reference type is not allowed in a union

[Explanation]

A member with reference type is not allowed in a union.

E4935B	"typedef" may not be specified here

[Explanation]

"typedef" may not be specified here.

E4936B	redeclaration of "entity" alters its access

[Explanation]

Redeclaration of "entity" alters its access.

E4937B	a class or namespace qualified name is required

[Explanation]

A class or namespace qualified name is required.

E4938B	return type "int" omitted in declaration of function "main"

[Explanation]

Return type "int" omitted in declaration of function "main".

E4939B	pointer-to-member representation "xxxx" is too restrictive for "entity"

[Explanation]

Pointer-to-member representation "xxxx" is too restrictive for "entity".

E4940B	missing return statement at end of non-void function "entity"

[Explanation]

Missing return statement at end of non-void function "entity".

E4941B	duplicate using-declaration of "entity" ignored

[Explanation]

Duplicate using-declaration of "entity" ignored.

E4946B	name following "template" must be a member template

[Explanation]

Name following "template" must be a member template.

E4947B	name following "template" must have a template argument list

[Explanation]

Name following "template" must have a template argument list.

E4948B	nonstandard local-class friend declaration -- no prior declaration in the enclosing scope

[Explanation]

Nonstandard local-class friend declaration -- no prior declaration in the enclosing scope.

E4949B	specifying a default argument when redeclaring an unreferenced function template is nonstandard

[Explanation]

Specifying a default argument when redeclaring an unreferenced function template is nonstandard.

E4951B	return type of function "main" must be "int"

[Explanation]

Return type of function "main" must be "int".

E4952B	a template parameter may not have class type

[Explanation]

A template parameter may not have class type.

E4953B	a default template argument cannot be specified on the declaration of a member of a class template

[Explanation]

A default template argument cannot be specified on the declaration of a member of a class template.

E4954B	a return statement is not allowed in a handler of a function try block of a constructor

[Explanation]

A return statement is not allowed in a handler of a function try block of a constructor.

E4955B	ordinary and extended designators cannot be combined in an initializer designation

[Explanation]

Ordinary and extended designators cannot be combined in an initializer designation.

E4956B	the second subscript must not be smaller than the first

[Explanation]

The second subscript must not be smaller than the first.

E4960B	type used as constructor name does not match type "type"

[Explanation]

Type used as constructor name does not match type "type".

E4961B	use of a type with no linkage to declare a variable with linkage

[Explanation]

Use of a type with no linkage to declare a variable with linkage.

E4962B	use of a type with no linkage to declare a function

[Explanation]

Use of a type with no linkage to declare a function.

E4963B	return type may not be specified on a constructor

[Explanation]

Return type may not be specified on a constructor.

E4964B	return type may not be specified on a destructor

[Explanation]

Return type may not be specified on a destructor.

E4965B	incorrectly formed universal character name

[Explanation]

Incorrectly formed universal character name.

E4966B	universal character name specifies an invalid character

[Explanation]

Universal character name specifies an invalid character.

E4967B	a universal character name cannot designate a character in the basic character set

[Explanation]

A universal character name cannot designate a character in the basic character set.

E4968B	this universal character is not allowed in an identifier

[Explanation]

This universal character is not allowed in an identifier.

E4969B	the identifier __VA_ARGS__ can only appear in the replacement lists of variadic macros

[Explanation]

The identifier __VA_ARGS__ can only appear in the replacement lists of variadic macros.

E4971B	array range designators cannot be applied to dynamic initializers

[Explanation]

Array range designators cannot be applied to dynamic initializers.

E4972B	property name cannot appear here

[Explanation]

Property name cannot appear here.

E4975B	a variable-length array type is not allowed

[Explanation]

A variable-length array type is not allowed.

E4976B	a compound literal is not allowed in an integral constant expression

[Explanation]

A compound literal is not allowed in an integral constant expression.

E4977B	a compound literal of type "type" is not allowed

[Explanation]

A compound literal of type "type" is not allowed.

E4978B	a template friend declaration cannot be declared in a local class

[Explanation]

A template friend declaration cannot be declared in a local class.

E4979B	ambiguous "?" operation: second operand of type "type" can be converted to third operand type "type", and vice versa

[Explanation]

Ambiguous "?" operation: Second operand of type "type" can be converted to third operand type "type", and vice versa.

E4980B	call of an object of a class type without appropriate operator() or conversion functions to pointer-to-function type

[Explanation]

Call of an object of a class type without appropriate operator() or conversion functions to pointer-to-function type.

E4982B	there is more than one way an object of type "type" can be called for the argument list:

[Explanation]

There is more than one way an object of type "type" can be called for the argument list.

E4983B	loop in sequence of "operator->" functions starting at class "type"

[Explanation]

Loop in sequence of "operator->" functions starting at class "type".

E4984B	typedef name has already been declared (with similar type)

[Explanation]

Typedef name has already been declared (with similar type).

E4985B	operator new and operator delete cannot be given internal linkage

[Explanation]

Operator new and operator delete cannot be given internal linkage.

E4986B	storage class "mutable" is not allowed for anonymous unions

[Explanation]

Storage class "mutable" is not allowed for anonymous unions.

E4988B	abstract class type %t is not allowed as catch type:

[Explanation]

Abstract class type %t is not allowed as catch type.

E4989B	a qualified function type cannot be used to declare a nonmember function or a static member function

[Explanation]

A qualified function type cannot be used to declare a nonmember function or a static member function.

E4990B	a qualified function type cannot be used to declare a parameter

[Explanation]

A qualified function type cannot be used to declare a parameter.

E4991B	cannot create a pointer or reference to qualified function type

[Explanation]

Cannot create a pointer or reference to qualified function type.

E4992B	extra braces are nonstandard

[Explanation]

Extra braces are nonstandard.

E4994B	subtraction of pointer types %t1 and %t2 is nonstandard

[Explanation]

Subtraction of pointer types %t1 and %t2 is nonstandard.

E4995B	an empty template parameter list is not allowed in a template template parameter declaration

[Explanation]

An empty template parameter list is not allowed in a template template parameter declaration.

E4996B	expected "class"

[Explanation]

Eexpected "class".

E4997B	the "class" keyword must be used when declaring a template template parameter

[Explanation]

The "class" keyword must be used when declaring a template template parameter.

E4999B	a qualified name is not allowed for a friend declaration that is a function definition

[Explanation]

A qualified name is not allowed for a friend declaration that is a function definition.

E5000B	%n1 is not compatible with %n2

[Explanation]

%n1 is not compatible with %n2.

E5001B	a storage class may not be specified here

[Explanation]

A storage class may not be specified here.

E5002B	class member designated by a using-declaration must be visible in a direct base class

[Explanation]

Class member designated by a using-declaration must be visible in a direct base class.

E5007B	a template template parameter cannot have the same name as one of its template parameters

[Explanation]

A template template parameter cannot have the same name as one of its template parameters.

E5008B	recursive instantiation of default argument

[Explanation]

Recursive instantiation of default argument.

E5009B	a parameter of a template template parameter cannot depend on the type of another template parameter

[Explanation]

A parameter of a template template parameter cannot depend on the type of another template parameter.

E5010B	%n is not an entity that can be defined

[Explanation]

%n is not an entity that can be defined.

E5011B	destructor name must be qualified

[Explanation]

Destructor name must be qualified.

E5012B	friend class name may not be introduced with "typename"

[Explanation]

Friend class name may not be introduced with "typename".

E5013B	a using-declaration may not name a constructor or destructor

[Explanation]

A using-declaration may not name a constructor or destructor.

E5014B	a qualified friend template declaration must refer to a specific previously declared template

[Explanation]

A qualified friend template declaration must refer to a specific previously declared template.

E5015B	invalid specifier in class template declaration

[Explanation]

Invalid specifier in class template declaration.

E5018B	%n has no member class %sq

[Explanation]

%n has no member class %sq.

E5019B	the global scope has no class named %sq

[Explanation]

The global scope has no class named %sq.

E5020B	recursive instantiation of template default argument

[Explanation]

Recursive instantiation of template default argument.

E5021B	access declarations and using-declarations cannot appear in unions

[Explanation]

Access declarations and using-declarations cannot appear in unions.

E5022B	%no is not a class member

[Explanation]

%no is not a class member.

E5023B	nonstandard member constant declaration is not allowed

[Explanation]

Nonstandard member constant declaration is not allowed.

E5028B	invalid redeclaration of nested class

[Explanation]

Invalid redeclaration of nested class.

E5029B	type containing an unknown-size array is not allowed

[Explanation]

Type containing an unknown-size array is not allowed.

E5030B	a variable with static storage duration cannot be defined within an inline function

[Explanation]

A variable with static storage duration cannot be defined within an inline function.

E5031B	an entity with internal linkage cannot be referenced within an inline function with external linkage

[Explanation]

An entity with internal linkage cannot be referenced within an inline function with external linkage.

E5032B	argument type %t does not match this type-generic function macro

[Explanation]

Argument type %t does not match this type-generic function macro.

E5034B	friend declaration cannot add default arguments to previous declaration

[Explanation]

Friend declaration cannot add default arguments to previous declaration.

E5035B	%n cannot be declared in this scope

[Explanation]

%n cannot be declared in this scope.

E5036B	the reserved identifier %sq may only be used inside a function

[Explanation]

The reserved identifier %sq may only be used inside a function.

E5037B	this universal character cannot begin an identifier

[Explanation]

This universal character cannot begin an identifier.

E5038B	expected a string literal

[Explanation]

Expected a string literal.

E5039B	unrecognized STDC pragma

[Explanation]

Unrecognized STDC pragma.

E5040B	expected "ON", "OFF", or "DEFAULT"

[Explanation]

Expected "ON", "OFF", or "DEFAULT".

E5041B	a STDC pragma may only appear between declarations in the global scope or before any statements or declarations in a block scope

[Explanation]

A STDC pragma may only appear between declarations in the global scope or before any statements or declarations in a block scope.

E5042B	incorrect use of va_copy

[Explanation]

Incorrect use of va_copy.

E5043B	%s can only be used with floating-point types

[Explanation]

%s can only be used with floating-point types.

E5044B	complex type is not allowed

[Explanation]

Complex type is not allowed.

E5045B	invalid designator kind

[Explanation]

Invalid designator kind.

E5047B	complex floating-point operation result is out of range

[Explanation]

Complex floating-point operation result is out of range.

E5049B	an initializer cannot be specified for a flexible array member

[Explanation]

An initializer cannot be specified for a flexible array member.

E5051B	standard requires that %n be given a type by a subsequent declaration ("int" assumed)

[Explanation]

Standard requires that %n be given a type by a subsequent declaration ("int" assumed).

E5052B	a definition is required for inline %n

[Explanation]

A definition is required for inline %n.

E5054B	a floating-point type must be included in the type specifier for a _Complex or _Imaginary type

[Explanation]

A floating-point type must be included in the type specifier for a _Complex or _Imaginary type.

E5055B	types cannot be declared in anonymous unions

[Explanation]

Types cannot be declared in anonymous unions.

E5061B	declaration of %n is incompatible with a declaration in another translation unit

[Explanation]

Declaration of %n is incompatible with a declaration in another translation unit.

E5062B	the other declaration is %p

[Explanation]

The other declaration is %p.

E5063B	detected during compilation of secondary translation unit %sq

[Explanation]

Detected during compilation of secondary translation unit %sq.

E5064B	compilation of secondary translation unit %sq

[Explanation]

Compilation of secondary translation unit %sq.

E5065B	a field declaration cannot have a type involving a variable length array

[Explanation]

A field declaration cannot have a type involving a variable length array.

E5066B	declaration of %n had a different meaning during compilation of %sq

[Explanation]

Declaration of %n had a different meaning during compilation of %sq.

E5067B	expected "template"

[Explanation]

Expected "template".

E5068B	"export" cannot be used on an explicit instantiation

[Explanation]

"export" cannot be used on an explicit instantiation.

E5069B	"export" cannot be used on this declaration

[Explanation]

"export" cannot be used on this declaration.

E5070B	a member of an unnamed namespace cannot be declared "export"

[Explanation]

A member of an unnamed namespace cannot be declared "export".

E5071B	a template cannot be declared "export" after it has been defined

[Explanation]

A template cannot be declared "export" after it has been defined.

E5072B	a declaration cannot have a label

[Explanation]

A declaration cannot have a label.

E5073B	support for exported templates is disabled

[Explanation]

Support for exported templates is disabled.

E5075B	%n already defined during compilation of %sq

[Explanation]

%n already defined during compilation of %sq.

E5076B	%n already defined in another translation unit

[Explanation]

%n already defined in another translation unit.

E5077B	a non-static local variable may not be used in a __based specification

[Explanation]

A non-static local variable may not be used in a __based specification.

E5081B	a field with the same name as its class cannot be declared in a class with a user-declared constructor

[Explanation]

A field with the same name as its class cannot be declared in a class with a user-declared constructor.

E5084B	%n cannot be instantiated -- it has been explicitly specialized in the translation unit containing the exported definition

[Explanation]

%n cannot be instantiated -- it has been explicitly specialized in the translation unit containing the exported definition.

E5085B	object type is: %s

[Explanation]

Object type is: %s.

E5086B	the object has cv-qualifiers that are not compatible with the member %n

[Explanation]

The object has cv-qualifiers that are not compatible with the member %n.

E5087B	no instance of %n matches the argument list and object (the object has cv-qualifiers that prevent a match)

[Explanation]

No instance of %n matches the argument list and object (the object has cv-qualifiers that prevent a match).

E5088B	an attribute specifies a mode incompatible with %t

[Explanation]

An attribute specifies a mode incompatible with %t.

E5089B	there is no type with the width specified

[Explanation]

There is no type with the width specified.

E5090B	invalid alignment value specified by attribute

[Explanation]

Invalid alignment value specified by attribute.

E5091B	invalid attribute for %t

[Explanation]

Invalid attribute for %t.

E5092B	invalid attribute for %n

[Explanation]

Invalid attribute for %n.

E5094B	attribute %sq does not take arguments

[Explanation]

Attribute %sq does not take arguments.

E5096B	expected an attribute name

[Explanation]

Expected an attribute name.

E5098B	attributes may not appear here

[Explanation]

Attributes may not appear here.

E5099B	invalid argument to attribute %sq

[Explanation]

Invalid argument to attribute %sq.

E5102B	"goto *expr" is nonstandard

[Explanation]

"goto *expr" is nonstandard.

E5103B	taking the address of a label is nonstandard

[Explanation]

Taking the address of a label is nonstandard.

E5106B	attribute %sq is only allowed in a function definition

[Explanation]

Attribute %sq is only allowed in a function definition.

E5107B	the "transparent_union" attribute only applies to unions, and %t is not a union

[Explanation]

The "transparent_union" attribute only applies to unions, and %t is not a union.

E5111B	only parameters can be transparent

[Explanation]

Only parameters can be transparent.

E5112B	the %sq attribute does not apply to local variables

[Explanation]

The %sq attribute does not apply to local variables.

E5113B	attributes are not permitted in a function definition

[Explanation]

Attributes are not permitted in a function definition.

E5115B	the second constant in a case range must be larger than the first

[Explanation]

The second constant in a case range must be larger than the first.

E5116B	an asm name is not permitted in a function definition

[Explanation]

An asm name is not permitted in a function definition.

E5118B	unknown register name "%s"

[Explanation]

Unknown register name "%s".

E5120B	unknown asm constraint modifier '%s'

[Explanation]

Unknown asm constraint modifier '%s'.

E5121B	unknown asm constraint letter '%s'

[Explanation]

Unknown asm constraint letter '%s'.

E5122B	asm operand has no constraint letter

[Explanation]

Asm operand has no constraint letter.

E5123B	an asm output operand must have one of the '=' or '+' modifiers

[Explanation]

An asm output operand must have one of the '=' or '+' modifiers.

E5124B	an asm input operand may not have the '=' or '+' modifiers

[Explanation]

An asm input operand may not have the '=' or '+' modifiers.

E5125B	too many operands to asm statement (maximum is 10)

[Explanation]

Too many operands to asm statement (maximum is 10).

E5126B	too many colons in asm statement

[Explanation]

Too many colons in asm statement.

E5127B	register "%s" used more than once

[Explanation]

Register "%s" used more than once.

E5128B	register "%s" is both used and clobbered

[Explanation]

Register "%s" is both used and clobbered.

E5129B	register "%s" clobbered more than once

[Explanation]

Register "%s" clobbered more than once.

E5130B	register "%s" has a fixed purpose and may not be used in an asm statement

[Explanation]

Register "%s" has a fixed purpose and may not be used in an asm statement.

E5131B	register "%s" has a fixed purpose and may not be clobbered in an asm statement

[Explanation]

Register "%s" has a fixed purpose and may not be clobbered in an asm statement.

E5132B	an empty clobbers list must be omitted entirely

[Explanation]

An empty clobbers list must be omitted entirely.

E5133B	expected an asm operand

[Explanation]

Expected an asm operand.

E5134B	expected a register to clobber

[Explanation]

Expected a register to clobber.

E5135B	"format" attribute applied to %n which does not have variable arguments

[Explanation]

"format" attribute applied to %n which does not have variable arguments.

E5136B	first substitution argument is not the first variable argument

[Explanation]

First substitution argument is not the first variable argument.

E5137B	format argument index is greater than number of parameters

[Explanation]

Format argument index is greater than number of parameters.

E5138B	format argument does not have string type

[Explanation]

Format argument does not have string type.

E5139B	the "template" keyword used for syntactic disambiguation may only be used within a template

[Explanation]

The "template" keyword used for syntactic disambiguation may only be used within a template.

E5140B	a debug option must be specified on the command-line for the db_opt pragma to be used

[Explanation]

A debug option must be specified on the command-line for the db_opt pragma to be used.

E5144B	storage class must be auto or register

[Explanation]

Storage class must be auto or register.

E5145B	%t1 would have been promoted to %t2 when passed through the ellipsis parameter; use the latter type instead

[Explanation]

%t1 would have been promoted to %t2 when passed through the ellipsis parameter; use the latter type instead.

E5146B	%sq is not a base class member

[Explanation]

%sq is not a base class member.

E5147B	__super cannot appear after "::"

[Explanation]

__super cannot appear after "::".

E5148B	__super may only be used in a class scope

[Explanation]

__super may only be used in a class scope.

E5149B	__super must be followed by "::"

[Explanation]

__super must be followed by "::".

E5150B	[%s instantiation contexts not shown]

[Explanation]

[%s instantiation contexts not shown.]

E5153B	declaration does not match its alias %n

[Explanation]

Declaration does not match its alias %n.

E5154B	entity declared as alias cannot have definition

[Explanation]

Entity declared as alias cannot have definition.

E5158B	void return type cannot be qualified

[Explanation]

Void return type cannot be qualified.

E5161B	a member template corresponding to %no is declared as a template of a different kind in another translation unit

[Explanation]

A member template corresponding to %no is declared as a template of a different kind in another translation unit.

E5163B	va_start should only appear in a function with an ellipsis parameter

[Explanation]

va_start should only appear in a function with an ellipsis parameter.

E5166B	statement expressions are only allowed in block scope

[Explanation]

Statement expressions are only allowed in block scope.

E5167B	from translation unit

[Explanation]

From translation unit.

E5170B	unrecognized UPC pragma

[Explanation]

Unrecognized UPC pragma.

E5171B	shared block size does not match one previously specified

[Explanation]

Shared block size does not match one previously specified.

E5173B	the block size of a shared array must be greater than zero

[Explanation]

The block size of a shared array must be greater than zero.

E5174B	multiple block sizes not allowed

[Explanation]

Multiple block sizes not allowed.

E5175B	strict or relaxed requires shared

[Explanation]

Strict or relaxed requires shared.

E5176B	THREADS not allowed in this context

[Explanation]

THREADS not allowed in this context.

E5177B	block size specified exceeds the maximum value of %s

[Explanation]

Block size specified exceeds the maximum value of %s.

E5178B	function returning shared is not allowed

[Explanation]

Function returning shared is not allowed.

E5179B	only arrays of a shared type can be dimensioned to a multiple of THREADS

[Explanation]

Only arrays of a shared type can be dimensioned to a multiple of THREADS.

E5180B	one dimension of an array of a shared type must be a multiple of THREADS when the number of threads is nonconstant

[Explanation]

One dimension of an array of a shared type must be a multiple of THREADS when the number of threads is nonconstant.

E5181B	shared type inside a struct or union is not allowed

[Explanation]

Shared type inside a struct or union is not allowed.

E5182B	parameters may not have shared types

[Explanation]

Parameters may not have shared types.

E5183B	a dynamic THREADS dimension requires a definite block size

[Explanation]

A dynamic THREADS dimension requires a definite block size.

E5184B	shared variables must be static or extern

[Explanation]

Shared variables must be static or extern.

E5187B	branching into or out of a upc_forall loop is not allowed

[Explanation]

Branching into or out of a upc_forall loop is not allowed.

E5188B	affinity expression must have a shared type or point to a shared type

[Explanation]

Affinity expression must have a shared type or point to a shared type.

E5190B	shared void* types can only be compared for equality

[Explanation]

Shared void* types can only be compared for equality.

E5196B	the hidden declaration is %p

[Explanation]

The hidden declaration is %p.

E5199B	%npd must have external C linkage

[Explanation]

%npd must have external C linkage.

E5201B	typedef %sq may not be used in an elaborated type specifier

[Explanation]

Typedef %sq may not be used in an elaborated type specifier.

E5203B	parameter %sq may not be redeclared in a catch clause of function try block

[Explanation]

Parameter %sq may not be redeclared in a catch clause of function try block.

E5204B	the initial explicit specialization of %n must be declared in the namespace containing the template

[Explanation]

The initial explicit specialization of %n must be declared in the namespace containing the template.

E5206B	"template" must be followed by an identifier

[Explanation]

"template" must be followed by an identifier.

E5207B	MYTHREAD not allowed in this context

[Explanation]

MYTHREAD not allowed in this context.

E5208B	layout qualifier cannot qualify pointer to shared

[Explanation]

Layout qualifier cannot qualify pointer to shared.

E5209B	layout qualifier cannot qualify an incomplete array

[Explanation]

Layout qualifier cannot qualify an incomplete array.

E5210B	declaration of %sq hides handler parameter

[Explanation]

Declaration of %sq hides handler parameter.

E5212B	this pragma cannot be used in a _Pragma operator (a #pragma directive must be used)

[Explanation]

This pragma cannot be used in a _Pragma operator (a #pragma directive must be used).

E5216B	an asm name is not allowed on a nonstatic member declaration

[Explanation]

An asm name is not allowed on a nonstatic member declaration.

E5219B	the "init_priority" attribute can only be used for namespace scope variables of class types

[Explanation]

The "init_priority" attribute can only be used for namespace scope variables of class types.

E5226B	labels can be referenced only in function definitions

[Explanation]

Labels can be referenced only in function definitions.

E5227B	transfer of control into a statement expression is not allowed

[Explanation]

Transfer of control into a statement expression is not allowed.

E5228B	transfer of control out of a statement expression is not allowed

[Explanation]

Transfer of control out of a statement expression is not allowed.

E5229B	this statement is not allowed inside of a statement expression

[Explanation]

This statement is not allowed inside of a statement expression.

E5230B	a non-POD class definition is not allowed inside of a statement expression

[Explanation]

A non-POD class definition is not allowed inside of a statement expression.

E5231B	destructible entities are not allowed inside of a statement expression

[Explanation]

Destructible entities are not allowed inside of a statement expression.

E5232B	a dynamically-initialized local static variable is not allowed inside of a statement expression

[Explanation]

A dynamically-initialized local static variable is not allowed inside of a statement expression.

E5233B	a variable-length array is not allowed inside of a statement expression

[Explanation]

A variable-length array is not allowed inside of a statement expression.

E5234B	a statement expression is not allowed inside of a default argument

[Explanation]

A statement expression is not allowed inside of a default argument.

E5236B	interface types cannot have virtual base classes

[Explanation]

Interface types cannot have virtual base classes.

E5237B	interface types cannot specify "private" or "protected"

[Explanation]

Interface types cannot specify "private" or "protected".

E5238B	interface types can only derive from other interface types

[Explanation]

Interface types can only derive from other interface types.

E5239B	"type" is an interface type

[Explanation]

"type" is an interface type.

E5240B	interface types cannot have typedef members

[Explanation]

Interface types cannot have typedef members.

E5241B	interface types cannot have user-declared constructors or destructors

[Explanation]

Interface types cannot have user-declared constructors or destructors.

E5242B	interface types cannot have user-declared member operators

[Explanation]

Interface types cannot have user-declared member operators.

E5243B	interface types cannot be declared in functions

[Explanation]

Interface types cannot be declared in functions.

E5244B	cannot declare interface templates

[Explanation]

Cannot declare interface templates.

E5245B	interface types cannot have data members

[Explanation]

Interface types cannot have data members.

E5246B	interface types cannot contain friend declaration

[Explanation]

Interface types cannot contain friend declaration.

E5247B	interface types cannot have nested classes

[Explanation]

Interface types cannot have nested classes.

E5248B	interface types cannot be nested class types

[Explanation]

Interface types cannot be nested class types.

E5249B	interface types cannot have member templates

[Explanation]

Interface types cannot have member templates.

E5250B	interface types cannot have static member functions

[Explanation]

Interface types cannot have static member functions.

E5251B	this pragma cannot be used in a __pragma operator (a #pragma directive must be used)

[Explanation]

This pragma cannot be used in a __pragma operator (a #pragma directive must be used).

E5252B	qualifier must be base class of "type"

[Explanation]

Qualifier must be base class of "type".

E5253B	declaration must correspond to a pure virtual member function in the indicated base class

[Explanation]

Declaration must correspond to a pure virtual member function in the indicated base class.

E5254B	integer overflow in internal computation due to size or complexity of "type"

[Explanation]

Integer overflow in internal computation due to size or complexity of "type".

E5255B	integer overflow in internal computation

[Explanation]

Integer overflow in internal computation.

E5256B	__w64 can only be specified on int, long, and pointer types

[Explanation]

__w64 can only be specified on int, long, and pointer types.

E5260B	invalid alignment specifier value

[Explanation]

Invalid alignment specifier value.

E5261B	expected an integer literal

[Explanation]

Expected an integer literal.

E5263B	expected an argument value for the "xxxx" attribute parameter

[Explanation]

Expected an argument value for the "xxxx" attribute parameter.

E5264B	invalid argument value for the "xxxx" attribute parameter

[Explanation]

Invalid argument value for the "xxxx" attribute parameter.

E5265B	expected a boolean value for the "xxxx" attribute parameter

[Explanation]

Expected a boolean value for the "xxxx" attribute parameter.

E5266B	a positional argument cannot follow a named argument in an attribute

[Explanation]

A positional argument cannot follow a named argument in an attribute.

E5267B	attribute "xxxx" has no parameter named "xxxx"

[Explanation]

Attribute "xxxx" has no parameter named "xxxx".

E5268B	expected an argument list for the "xxxx" attribute

[Explanation]

Expected an argument list for the "xxxx" attribute.

E5269B	expected a "," or "]"

[Explanation]

Expected a "," or "]".

E5270B	attribute argument "xxxx" has already been given a value

[Explanation]

Attribute argument "xxxx" has already been given a value.

E5271B	a value cannot be assigned to the "xxxx" attribute

[Explanation]

A value cannot be assigned to the "xxxx" attribute.

E5272B	a throw expression may not have pointer-to-incomplete type

[Explanation]

A throw expression may not have pointer-to-incomplete type.

E5273B	alignment-of operator applied to incomplete type

[Explanation]

Alignment-of operator applied to incomplete type.

E5274B	"xxxx" may only be used as a standalone attribute

[Explanation]

"xxxx" may only be used as a standalone attribute.

E5275B	"xxxx" attribute cannot be used here

[Explanation]

"xxxx" attribute cannot be used here.

E5276B	unrecognized attribute "xxxx"

[Explanation]

Unrecognized attribute "xxxx".

E5277B	attributes are not allowed here

[Explanation]

Attributes are not allowed here.

E5278B	invalid argument value for the "xxxx" attribute parameter

[Explanation]

Invalid argument value for the "xxxx" attribute parameter.

E5279B	too many attribute arguments

[Explanation]

Too many attribute arguments.

E5280B	conversion from inaccessible base class "type" is not allowed

[Explanation]

Conversion from inaccessible base class "type" is not allowed.

E6001B	bad argument for #pragma int_to_unsigned

[Explanation]

Bad argument for #pragma int_to_unsigned.

E6002B	argument for #pragma int_to_unsigned must return an unsigned type

[Explanation]

Argument for #pragma int_to_unsigned must return an unsigned type.

E6003B	cannot preprocess encrypted file: "xxxx"

[Explanation]

Cannot preprocess encrypted file: "xxxx".

E6007B	__interrupt is specified

[Explanation]

__interrupt is specified.

E6010B	__io is specified

[Explanation]

__io is specified.

E6013B	entity-kind "entity" may not be initialized for __io

[Explanation]

Entity-kind "entity" may not be initialized for __io.

E6014B	#pragma xxxx: syntax error: unknown specifier

[Explanation]

#pragma section or #pragma segment: syntax error: unknown specifier.

E6015B	#pragma xxxx: invalid section name specified

[Explanation]

#pragma section or #pragma segment: invalid section name specified.

E6017B	#pragma xxxx: invalid section attr specified

[Explanation]

#pragma section or #pragma segment: invalid section attr specified.

E6018B	#pragma xxxx: syntax error: address is expected

[Explanation]

#pragma section or #pragma segment: syntax error: address is expected.

E6019B	#pragma xxxx: address is not integral constant expression

[Explanation]

#pragma section or #pragma segment: address is not integral constant expression.

E6020B	#pragma inline: syntax error: unknown specifier

[Explanation]

#pragma inline: syntax error: unknown specifier.

E6021B	#pragma intvect: syntax error: unknown specifier

[Explanation]

#pragma intvect: syntax error: unknown specifier.

E6022B	#pragma intvect: syntax error: vector number is expected

[Explanation]

#pragma intvect: syntax error: vector number is expected.

E6023B	#pragma intvect: vector number is not integral constant expression

[Explanation]

#pragma intvect: vector number is not integral constant expression.

E6024B	#pragma intvect: same vector number exist

[Explanation]

#pragma intvect: same vector number exist.

E6025B	#pragma intvect: invalid type of interrupt function

[Explanation]

#pragma intvect: invalid type of interrupt function.

E6026B	#pragma intvect: interrupt function is not found

[Explanation]

#pragma intvect: interrupt function is not found.

E6027B	#pragma defvect: syntax error: unknown specifier

[Explanation]

#pragma defvect: syntax error: unknown specifier.

E6028B	#pragma defvect: duplicate defvect function

[Explanation]

#pragma defvect: duplicate defvect function.

E6029B	#pragma defvect: invalid type of interrupt function

[Explanation]

#pragma defvect: invalid type of interrupt function.

E6030B	#pragma defvect: interrupt function is not found

[Explanation]

#pragma defvect: interrupt function is not found.

E6031B	#pragma ilm: syntax error: interrupt level is expected

[Explanation]

#pragma ilm: syntax error: interrupt level is expected.

E6033B	#pragma ilm: invalid constant for interrupt level value

[Explanation]

#pragma ilm: invalid constant for interrupt level value.

E6034B	#pragma ilm: interrupt level is out of range

[Explanation]

#pragma ilm: interrupt level is out of range.

E6035B	#pragma noilm: `#pragma ilm' not exist

[Explanation]

#pragma noilm: `#pragma ilm' not exist.

E6036B	`#pragma noilm' expected

[Explanation]

`#pragma noilm' expected.

E6037B	`#pragma endasm' expected

[Explanation]

`#pragma endasm' expected.

E6050B	#pragma loop unroll: unroll count number is out of range

[Explanation]

#pragma loop unroll: unroll count number is out of range.

E6051B	#pragma statement if: branch rate number is out of range

[Explanation]

#pragma statement if: branch rate number is out of range.

E6052B	#pragma intvect: vector number is out of range

[Explanation]

#pragma intvect: vector number is out of range.

E6055B	argument xxxx of xxxx should be immediately a value

[Explanation]

Argument xxxx of xxxx should be immediately a value.

E6057B	argument xxxx of xxxx should be an accumulator number defined by media.h

[Explanation]

Argument xxxx of xxxx should be an accumulator number defined by media.h.

F9001B	#include file "xxxx" includes itself

[Explanation]

#include file "xxxx" includes itself.

F9002B	out of memory

[Explanation]

Out of memory.

F9003B	could not open source file "xxxx"

[Explanation]

Could not open source file "xxxx".

F9004B	expected a file name

[Explanation]

Expected a file name.

F9005B	"xxxx" is not a valid source file name

[Explanation]

"xxxx" is not a valid source file name.

F9006B	#error directive: xxxx

[Explanation]

#error directive: xxxx

F9007B	program too large or complicated to compile

[Explanation]

Program too large or complicated to compile.

F9008B	could not open temporary file "xxxx"

[Explanation]

Could not open temporary file "xxxx".

F9009B	name of directory for temporary files is too long ("xxxx")

[Explanation]

Name of directory for temporary files is too long ("xxxx").

F9010B	could not open source file "xxxx" (no directories in search list)

[Explanation]

Could not open source file "xxxx" (no directories in search list).

F9011B	error while writing xxxx file

[Explanation]

Error while writing xxxx file.

F9012B	invalid intermediate language file

[Explanation]

Invalid intermediate language file.

F9013B	error while deleting file "xxxx"

[Explanation]

Error while deleting file "xxxx".

F9014B	could not create instantiation request file "xxxx"

[Explanation]

Could not create instantiation request file "xxxx".

F9015B	unable to obtain mapped memory

[Explanation]

Unable to obtain mapped memory.

F9016B	insufficient memory for PCH memory allocation

[Explanation]

Insufficient memory for PCH memory allocation.

F9017B	"xxxx" is not a valid directory

[Explanation]

"xxxx" is not a valid directory.

F9018B	cannot build temporary file name

[Explanation]

Cannot build temporary file name.

F9019B	could not set locale "xxxx" to allow processing of multibyte characters

[Explanation]

Could not set locale "xxxx" to allow processing of multibyte characters.

F9020B	invalid output file: "xxxx"

[Explanation]

Invalid output file: "xxxx".

F9021B	cannot open output file: "xxxx"

[Explanation]

Cannot open output file: "xxxx".

F9022B	cannot open definition list file: "xxxx"

[Explanation]

Cannot open definition list file: "xxxx".

F9023B	invalid message file

[Explanation]

Invalid message file.

F9024B	invalid precompiled header file

[Explanation]

Invalid precompiled header file.

F9025B	cannot open exported template file: "xxxx"

[Explanation]

Cannot open exported template file: "xxxx".

F9026B	exported template file "xxxx" is corrupted

[Explanation]

Exported template file "xxxx" is corrupted.

F9027B	mangled name is too long

[Explanation]

Mangled name is too long.

F9028B	invalid export information file "xxxx" at line number "xxxx"

[Explanation]

Invalid export information file "xxxx" at line number "xxxx".

F9029B	not support asm statement in function with covariant return type

[Explanation]

Not support asm statement in function with covariant return type.

F9030B	multiple global functions named xxxx.

[Explanation]

Multiple global functions named xxxx.

F9031B	xxxx not supported by -Kcrossfile.

[Explanation]

xxxx not supported by -Kcrossfile.

F9032B	-Kpu requires =filename.

[Explanation]

-Kpu requires =filename.

F9033B	specified profile has invalid data

[Explanation]

Specified profile has invalid data.

F9099B [561]	invalid macro definition

[Explanation]

Invalid macro definition.

F9099B [562]	invalid macro undefinition

[Explanation]

Invalid macro undefinition.

F9099B [563]	invalid preprocessor output file

[Explanation]

Invalid preprocessor output file.

F9099B [564]	cannot open preprocessor output file

[Explanation]

Cannot open preprocessor output file.

F9099B [565]	IL file name must be specified if input is

[Explanation]

IL file name must be specified if input is.

F9099B [566]	invalid IL output file

[Explanation]

Invalid IL output file.

F9099B [567]	cannot open IL output file

[Explanation]

Cannot open IL output file.

F9099B [568]	invalid C output file

[Explanation]

Invalid C output file.

F9099B [569]	cannot open C output file

[Explanation]

Cannot open C output file.

F9099B [570]	error in debug option argument

[Explanation]

Error in debug option argument.

F9099B [571]	invalid option

[Explanation]

Invalid option.

F9099B [572]	back end requires name of IL file

[Explanation]

Back end requires name of IL file.

F9099B [573]	could not open IL file

[Explanation]

Could not open IL file.

F9099B [574]	invalid number

[Explanation]

Invalid number.

F9099B [575]	incorrect host CPU id

[Explanation]

Incorrect host CPU id.

F9099B [576]	invalid instantiation mode

[Explanation]

Invalid instantiation mode.

F9099B [578]	invalid error limit

[Explanation]

Invalid error limit.

F9099B [579]	invalid raw-listing output file

[Explanation]

Invalid raw-listing output file.

F9099B [580]	cannot open raw-listing output file

[Explanation]

Cannot open raw-listing output file.

F9099B [581]	invalid cross-reference output file

[Explanation]

Invalid cross-reference output file.

F9099B [582]	cannot open cross-reference output file

[Explanation]

Cannot open cross-reference output file.

F9099B [583]	invalid error output file

[Explanation]

Invalid error output file.

F9099B [584]	cannot open error output file

[Explanation]

Cannot open error output file.

F9099B [585]	virtual function tables can only be suppressed when compiling C++

[Explanation]

Virtual function tables can only be suppressed when compiling C++.

F9099B [586]	anachronism option can be used only when compiling C++

[Explanation]

Anachronism option can be used only when compiling C++.

F9099B [587]	instantiation mode option can be used only when compiling C++

[Explanation]

Instantiation mode option can be used only when compiling C++.

F9099B [588]	automatic instantiation mode can be used only when compiling C++

[Explanation]

Automatic instantiation mode can be used only when compiling C++.

F9099B [589]	implicit template inclusion mode can be used only when compiling C++

[Explanation]

Implicit template inclusion mode can be used only when compiling C++.

F9099B [590]	exception handling option can be used only when compiling C++

[Explanation]

Exception handling option can be used only when compiling C++.

F9099B [591]	strict ANSI mode is incompatible with K&R mode

[Explanation]

Strict ANSI mode is incompatible with K&R mode.

F9099B [592]	strict ANSI mode is incompatible with cfront mode

[Explanation]

Strict ANSI mode is incompatible with cfront mode.

F9099B [593]	missing source file name

[Explanation]

Missing source file name.

F9099B [594]	output files may not be specified when compiling several input files

[Explanation]

Output files may not be specified when compiling several input files.

F9099B [595]	too many arguments on command line

[Explanation]

Too many arguments on command line.

F9099B [596]	an output file was specified, but none is needed

[Explanation]

An output file was specified, but none is needed.

F9099B [597]	IL display requires name of IL file

[Explanation]

IL display requires name of IL file.

F9099B [600]	strict ANSI mode is incompatible with allowing anachronisms

[Explanation]

Strict ANSI mode is incompatible with allowing anachronisms.

F9099B [602]	local instantiation mode is incompatible with automatic instantiation

[Explanation]

Local instantiation mode is incompatible with automatic instantiation.

F9099B [613]	invalid error tag in diagnostic control option

[Explanation]

Invalid error tag in diagnostic control option.

F9099B [614]	invalid error number in diagnostic control option

[Explanation]

Invalid error number in diagnostic control option.

F9099B [622]	invalid precompiled header output file

[Explanation]

Invalid precompiled header output file.

F9099B [623]	cannot open precompiled header output file

[Explanation]

Cannot open precompiled header output file.

F9099B [625]	cannot open precompiled header input file

[Explanation]

Cannot open precompiled header input file.

F9099B [635]	invalid PCH memory size

[Explanation]

Invalid PCH memory size.

F9099B [636]	PCH options must appear first in the command line

[Explanation]

PCH options must appear first in the command line.

F9099B [638]	precompiled header files may not be used when compiling several input files

[Explanation]

Precompiled header files may not be used when compiling several input files.

F9099B [648]	strict ANSI mode is incompatible with Microsoft mode

[Explanation]

Strict ANSI mode is incompatible with Microsoft mode.

F9099B [649]	cfront mode is incompatible with Microsoft mode

[Explanation]

Cfront mode is incompatible with Microsoft mode.

F9099B [659]	wchar_t keyword option can be used only when compiling C++

[Explanation]

wchar_t keyword option can be used only when compiling C++.

F9099B [660]	invalid packing alignment value

[Explanation]

Invalid packing alignment value.

F9099B [675]	SVR4 C compatibility option can be used only when compiling ANSI C

[Explanation]

SVR4 C compatibility option can be used only when compiling ANSI C.

F9099B [677]	strict ANSI mode is incompatible with SVR4 C mode

[Explanation]

Strict ANSI mode is incompatible with SVR4 C mode.

F9099B [680]	invalid PCH directory

[Explanation]

Invalid PCH directory.

F9099B [690]	RTTI option can be used only when compiling C++

[Explanation]

RTTI option can be used only when compiling C++.

F9099B [699]	bool option can be used only when compiling C++

[Explanation]

Bool option can be used only when compiling C++.

F9099B [712]	array new and delete option can be used only when compiling C++

[Explanation]

Array new and delete option can be used only when compiling C++.

F9099B [736]	namespaces option can be used only when compiling C++

[Explanation]

Namespaces option can be used only when compiling C++.

F9099B [762]	special_subscript_cost option can be used only when compiling C++

[Explanation]

Special_subscript_cost option can be used only when compiling C++.

F9099B [763]	typename option can be used only when compiling C++

[Explanation]

Typename option can be used only when compiling C++.

F9099B [764]	implicit typename option can be used only when compiling C++

[Explanation]

Implicit typename option can be used only when compiling C++.

F9099B [770]	option "explicit" can be used only when compiling C++

[Explanation]

Option "explicit" can be used only when compiling C++.

F9099B [778]	option to control the for-init scope can be used only when compiling C++

[Explanation]

Option to control the for-init scope can be used only when compiling C++.

F9099B [781]	option to control warnings on for-init differences can be used only when compiling C++

[Explanation]

Option to control warnings on for-init differences can be used only when compiling C++.

F9099B [798]	option "old_specializations" can be used only when compiling C++

[Explanation]

Option "old_specializations" can be used only when compiling C++.

F9099B [813]	option "implicit_extern_c_type_conversion" can be used only when compiling C++

[Explanation]

Option "implicit_extern_c_type_conversion" can be used only when compiling C++.

F9099B [814]	strict ANSI mode is incompatible with long preserving rules

[Explanation]

Strict ANSI mode is incompatible with long preserving rules.

F9099B [820]	option "extern_inline" can be used only when compiling C++

[Explanation]

Option "extern_inline" can be used only when compiling C++.

F9099B [856]	option "guiding_decls" can be used only when compiling C++

[Explanation]

Option "guiding_decls" can be used only when compiling C++.

F9099B [874]	option "embedded_c++" can be used only when compiling C++

[Explanation]

Option "embedded_c++" can be used only when compiling C++.

F9099B [883]	invalid Microsoft version number

[Explanation]

Invalid Microsoft version number.

F9099B [889]	option "vla" can be used only when compiling C

[Explanation]

Option "vla" can be used only when compiling C.

F9099B [899]	option "enum_overloading" can be used only when compiling C++

[Explanation]

Option "enum_overloading" can be used only when compiling C++.

F9099B [903]	option "nonstd_qualifier_deduction" can be used only when compiling C++

[Explanation]

Option "nonstd_qualifier_deduction" can be used only when compiling C++.

F9099B [917]	invalid directory for instantiation files

[Explanation]

Invalid directory for instantiation files.

F9099B [918]	option "one_instantiation_per_object" can be used only when compiling C++

[Explanation]

Option "one_instantiation_per_object" can be used only when compiling C++.

F9099B [921]	an instantiation information file name may not be specified when compiling several input files

[Explanation]

An instantiation information file name may not be specified when compiling several input files.

F9099B [922]	option "one_instantiation_per_object" may not be used when compiling several input files

[Explanation]

Option "one_instantiation_per_object" may not be used when compiling several input files.

F9099B [923]	more than one command line option matches the abbreviation "--xxxx"

[Explanation]

More than one command line option matches the abbreviation "--xxxx".

F9099B [927]	late/early tiebreaker option can be used only when compiling C++

[Explanation]

Late/early tiebreaker option can be used only when compiling C++.

F9099B [931]	pending instantiations option can be used only when compiling C++

[Explanation]

Pending instantiations option can be used only when compiling C++.

F9099B [932]	invalid directory for #import files:

[Explanation]

invalid directory for #import files.

F9099B [933]	an import directory can be specified only in Microsoft mode

[Explanation]

An import directory can be specified only in Microsoft mode.

F9099B [943]	option "class_name_injection" can be used only when compiling C++

[Explanation]

Option "class_name_injection" can be used only when compiling C++.

F9099B [944]	option "arg_dep_lookup" can be used only when compiling C++

[Explanation]

Option "arg_dep_lookup" can be used only when compiling C++.

F9099B [945]	option "friend_injection" can be used only when compiling C++

[Explanation]

Option "friend_injection" can be used only when compiling C++.

F9099B [950]	option "nonstd_using_decl" can be used only when compiling C++

[Explanation]

Option "nonstd_using_decl" can be used only when compiling C++.

F9099B [957]	option "designators" can be used only when compiling C

[Explanation]

Option "designators" can be used only when compiling C.

F9099B [958]	option "extended_designators" can be used only when compiling C

[Explanation]

Option "extended_designators" can be used only when compiling C.

F9099B [974]	option "compound_literals" can be used only when compiling C

[Explanation]

Option "compound_literals" can be used only when compiling C.

F9099B [993]	invalid macro definition:

[Explanation]

Invalid macro definition.

F9099B [1004]	Sun mode is incompatible with cfront mode

[Explanation]

Sun mode is incompatible with cfront mode.

F9099B [1005]	strict ANSI mode is incompatible with Sun mode

[Explanation]

Strict ANSI mode is incompatible with Sun mode.

F9099B [1006]	Sun mode is only allowed when compiling C++

[Explanation]

Sun mode is only allowed when compiling C++.

F9099B [1017]	option "dep_name" can be used only when compiling C++

[Explanation]

Option "dep_name" can be used only when compiling C++.

F9099B [1024]	option "ignore_std" can be used only when compiling C++

[Explanation]

Option "ignore_std" can be used only when compiling C++.

F9099B [1025]	option "parse_templates" can be used only when compiling C++

[Explanation]

Option "parse_templates" can be used only when compiling C++.

F9099B [1026]	option "dep_name" cannot be used with "no_parse_templates"

[Explanation]

Option "dep_name" cannot be used with "no_parse_templates".

F9099B [1027]	language modes specified are incompatible

[Explanation]

Language modes specified are incompatible.

F9099B [1058]	option "export" can be used only when compiling C++

[Explanation]

Option "export" can be used only when compiling C++.

F9099B [1059]	option "export" cannot be used with "no_dep_name"

[Explanation]

Option "export" cannot be used with "no_dep_name".

F9099B [1060]	option "export" cannot be used with "implicit_include"

[Explanation]

Option "export" cannot be used with "implicit_include".

F9099B [1078]	the option to list makefile dependencies may not be specified when compiling more than one translation unit

[Explanation]

The option to list makefile dependencies may not be specified when compiling more than one translation unit.

F9099B [1080]	the option to generate preprocessed output may not be specified when compiling more than one translation unit

[Explanation]

The option to generate preprocessed output may not be specified when compiling more than one translation unit.

F9099B [1082]	"implicit_include" cannot be used when compiling more than one translation unit

[Explanation]

"implicit_include" cannot be used when compiling more than one translation unit.

F9099B [1104]	file name specified more than once:

[Explanation]

File name specified more than once.

F9099B [1141]	more than one preinclude option specified

[Explanation]

More than one preinclude option specified.

F9099B [1157]	unrecognized flag name

[Explanation]

Unrecognized flag name.

F9099B [1164]	the "short_enums" option is only valid in GNU C and GNU C++ modes

[Explanation]

The "short_enums" option is only valid in GNU C and GNU C++ modes.

F9099B [1191]	UPC mode is incompatible with C++ and K&R modes

[Explanation]

UPC mode is incompatible with C++ and K&R modes.

F9099B [1281]	option "export" requires distinct template signatures

[Explanation]

Option "export" requires distinct template signatures.

F9099B [9095]	invalid architecture specified by the --cpu option:

[Explanation]

Invalid architecture specified by the --cpu option.

F9099B [9096]	cannot open include files list output file

[Explanation]

Cannot open include files list output file.

F9099B [9097]	cannot open message file

[Explanation]

Cannot open message file.

F9099B [9098]	invalid assert definition

[Explanation]

Invalid assert definition.

APPENDIX D Reserved Pragma Directive

The pragma directive has been reserved by the C compiler is described.

■ Pragma directive has been reserved by C compiler

The following pragma directives have been reserved by the fcc911s command.

```
#pragma ARGSUSED
#pragma NOTREACHED
#pragma STDC
#pragma VARARGS
#pragma __printf_args
#pragma __scanf_args
#pragma builtin
#pragma can_instantiate
#pragma db_name
#pragma db_opt
#pragma define_type_info
#pragma diag_default
#pragma diag_error
#pragma diag_remark
#pragma diag_suppress
#pragma diag_warning
#pragma hdrstop
#pragma ident
#pragma int_to_unsigned
#pragma loop
#pragma no_pch
#pragma pcros
#pragma realos
#pragma statement
#pragma unknown_control_flow
#pragma weak
```

APPENDIX E About Reentrancy of C Library Functions

Reentrancy of C library functions is described.

■ About Reentrancy of C Library Functions

- List of reentrant functions

Table E-1 List of reentrant functions (1/2)

function name	header file name
abs	stdlib.h
atoi	stdlib.h
atol	stdlib.h
bsearch	stdlib.h
difftime	time.h
div	stdlib.h
isalnum	ctype.h
isalpha	ctype.h
iscntrl	ctype.h
isdigit	ctype.h
isgraph	ctype.h
islower	ctype.h
isprint	ctype.h
ispunct	ctype.h
isspace	ctype.h
isupper	ctype.h
isxdigit	ctype.h
labs	stdlib.h
ldiv	stdlib.h
memchr	string.h
memcmp	string.h
memcpy	string.h
memmove	string.h

Table E-1 List of reentrant functions (2/2)

function name	header file name
memset	string.h
qsort	stdlib.h
strcat	string.h
strchr	string.h
strcmp	string.h
strcpy	string.h
strcspn	string.h
strlen	string.h
strncat	string.h
strncmp	string.h
strncpy	string.h
strpbrk	string.h
strrchr	string.h
strspn	string.h
strstr	string.h
tolower	ctype.h
toupper	ctype.h
va_arg	stdarg.h
va_end	stdarg.h
va_start	stdarg.h
All runtime library functions	

- List of non-reentrant functions

Table E-2 List of non-reentrant functions (1/3)

function name	header file name
abort	stdlib.h
acos	math.h
asctime	time.h
asin	math.h
assert	assert.h
atan	math.h
atan2	math.h
atexit	stdlib.h
atof	stdlib.h
calloc	stdlib.h
ceil	math.h
clearerr	stdio.h
cos	math.h
cosh	math.h
ctime	time.h
exit	stdlib.h
exp	math.h
fabs	math.h
fclose	stdio.h
feof	stdio.h
ferror	stdio.h
fflush	stdio.h
fgetc	stdio.h
fgetpos	stdio.h
fgets	stdio.h
floor	math.h
fmod	math.h
fopen	stdio.h

Table E-2 List of non-reentrant functions (2/3)

function name	header file name
fprintf	stdio.h
fputc	stdio.h
fputs	stdio.h
fread	stdio.h
free	stdlib.h
freopen	stdio.h
frexp	math.h
fscanf	stdio.h
fseek	stdio.h
fsetpos	stdio.h
ftell	stdio.h
fwrite	stdio.h
getc	stdio.h
getchar	stdio.h
gets	stdio.h
gmtime	time.h
ldexp	math.h
localtime	time.h
log	math.h
log10	math.h
longjmp	setjmp.h
malloc	stdlib.h
mktime	time.h
modf	math.h
pow	math.h
printf	stdio.h
putc	stdio.h
putchar	stdio.h
puts	stdio.h
rand	stdlib.h
realloc	stdlib.h

Table E-2 List of non-reentrant functions (3/3)

function name	header file name
rewind	stdio.h
scanf	stdio.h
setbuf	stdio.h
setjmp	setjmp.h
setvbuf	stdio.h
sin	math.h
sinh	math.h
sprintf	stdio.h
sqrt	math.h
srand	stdlib.h
sscanf	stdio.h
stream_init	-
strftime	time.h
strtod	stdlib.h
strtok	string.h
strtol	stdlib.h
strtoul	stdlib.h
tan	math.h
tanh	math.h
ungetc	stdio.h
vfprintf	stdio.h
vprintf	stdio.h
vsprintf	stdio.h

INDEX

The index follows on the next page.
This is listed in alphabetic order.

Index

Symbols

A

B

C

T

U

W

CM81-00206-7E

FUJITSU SEMICONDUCTOR • CONTROLLER MANUAL

FR FAMILY

SOFTUNE ™ C/C++ COMPILER MANUAL

for V6

December 2011 the seventh edition

Published FUJITSU SEMICONDUCTOR LIMITED

Edited Sales Promotion Dept.