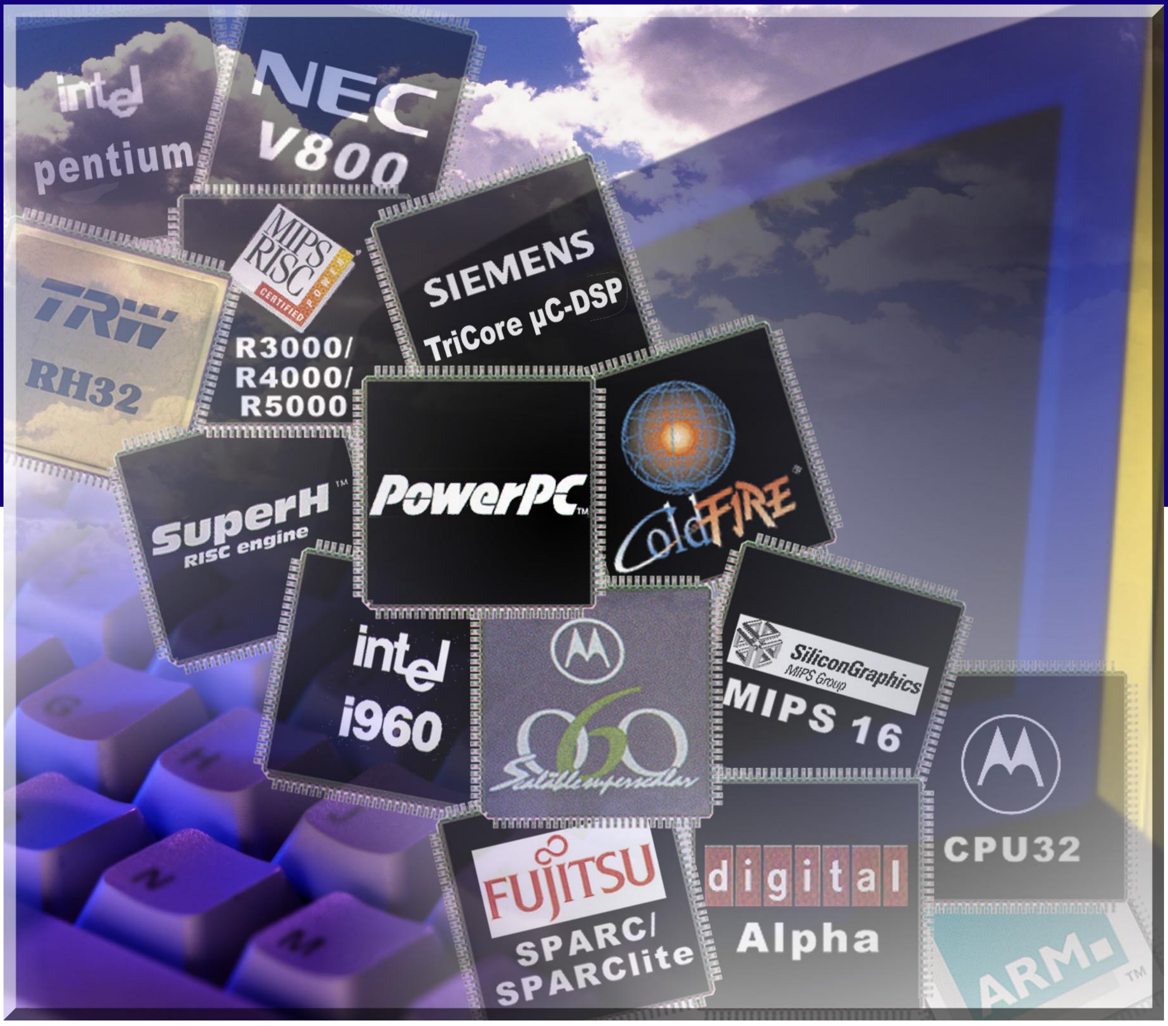
C • C++ • EC++ • ADA 95 • FORTRAN

OPTIMIZING COMPILERS



• Software Development Tools for Embedded Applications •

Green Hills

• SOFTWARE, INC. •

Optimizing C, C++, EC++, Ada 95, & FORTRAN Compilers

Green Hills' Optimizing C, C++, Embedded C++ (EC++), Ada 95, and FORTRAN Compilers consist of a Language-Specific Front-End, a Global Optimizer, and a Target-Specific Optimizer and Code Generator. All Green Hills compilers use the same Global Optimizer regardless of language or target, and the same Target-Specific Optimizers and Code Generators regardless of language. Optimizations may be optionally weighted for either maximum speed or minimum size.

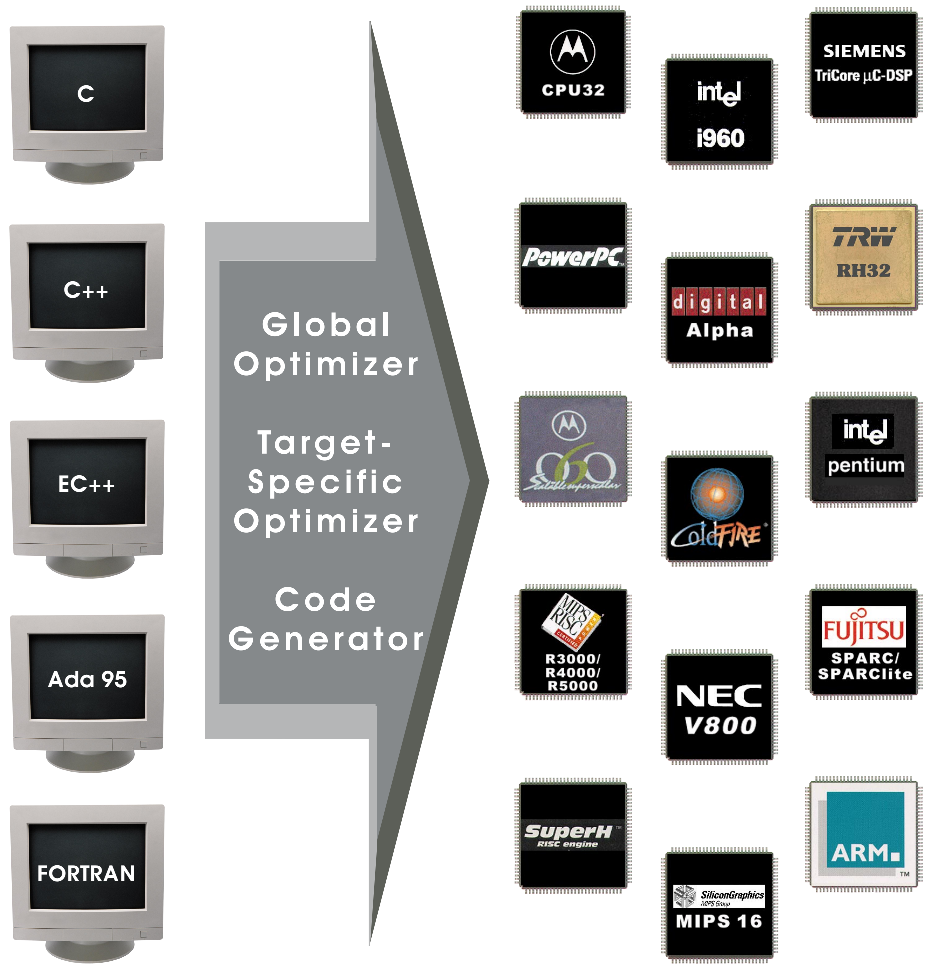
This modular approach to compiler building allows Green Hills Software to add support for new microprocessors quickly while maintaining a high degree of reliability.

The Green Hills Tool Chain consists of a Macro Assembler, Librarian, Linker and Utility Programs. The Macro Assembler assembles compiler-generated or user-coded assembly language files into object files. The Librarian manages object file libraries. The Linker links object files and object file libraries into executable programs.

Compiler Output

Green Hills' compilers can emit assembly language files for the Macro Assembler to assemble into ELF object files. For specific targets, they can emit ELF object files directly, which is significantly faster.

Green Hills' ELF compilers can emit DWARF 1.1 debug information for comprehensive source-level debugging in MULTI. COFF or BSD debug information is available for selected target processors.



Green Hills' Optimizing C, C++, EC++, Ada 95 and FORTRAN Compilers support 15 CPU families and use the same global optimizer and code generator. Code may be optimized for maximum speed or for minimum code size on a module by module basis.

Global Optimizer

Green Hills' Optimizing Compilers perform many standard global optimizations:

- Evaluate constant expressions
- Reduce constant multiples
- Reduce constant divides
- Reduce subscript strength
- Eliminate dead code
- Propagate constants, values, subexpressions, and machine state across blocks
- Optimize loop computations: remove loop invariants, cache lvalues in registers
- Allocate data by size and use to minimize wasted space
- Remove tail recursion
- Unroll loops

Green Hills' Optimizing Compilers also perform advanced global optimizations rarely found in competing compilers, including:

- Optimize loop control: use hardware loop counter
- Implement portable initialized global register variables and values
- Inline functions manually and automatically, across languages and compilations
- Register structs and unions
- Optimize function entry/exit
- Minimize code size
- Superscalar pipeline scheduling

Optimizations can significantly decrease program size and increase execution speed. Green Hills' compilers typically apply more than 100 optimizations to each program, producing fast code with a small footprint.

Peephole Optimizer

The Green Hills target-independent multi-pass Peephole Optimizer is the distilled descendant of 18 highly tuned target-specific peephole optimizers. It has more than 70 separate peephole optimizations, including:

- Renumber registers to delete moves
- Eliminate redundant loads and stores
- Multiply and add/sub => MADD/MSUB
- Combine instructions
- Use bit field extract/insert

Pipeline Scheduler

Modern pipelined machines have multicycle instructions that can overlap other instructions. The Green Hills multi-pass instruction Pipeline Scheduler maximizes throughput by overlapping multicycle instructions. It often achieves perfect instruction schedules.

Code Generator

The code generator for each target processor includes all Green Hills' target-specific optimizations:

- Expression tree reshaper
- Coloring register allocator
- Inline built-in functions
- Optimistic code generation

For any built-in function, the code generator can generate a specified inline instruction sequence.

Embedded Support

The Green Hills Embedded Features Package enhances the Compilers, Macro Assembler, Linker, Runtime Libraries, and Utility Programs to support embedded applications.

Compiler Enhancements

- ROMable code and data
- ROMable consts and strings
- PIC (Position Independent Code)
- PID (Position Independent Data)
- SDA (Small Data Area), also Zero Data Area and Tiny Data Area on some targets
- Section renaming
- Support for weak external symbols
- Built-in functions such as __EI (enable interrupts) and __DI (disable interrupts)
- Interrupt service routines in C
- Enhanced "asm" facility
- Efficient calling sequence

Macro Assembler Enhancements

- Support for weak externals
- Multiple sections
- Macros
- Absolute sections
- ORG statement
- Emits debug information so MULTI can debug the user's assembly file

Linker Enhancements

- Multiple sections
- Support for weak externals
- Comprehensive section map directives
- ROM/RAM section placement

Runtime Library Enhancements

- Startup routine to copy initial data values from ROM to RAM and clear BSS RAM
- Startup routine to relocate initialized pointers for PIC/PID
- Space-saving embedded versions of fopen, fclose, printf, scanf, etc.

Tool Chain

- The Green Hills Macro-Assembler has many embedded features such as support for weak symbols and section renaming control.
- The Green Hills Librarian is compatible with the Unix ar librarian.
- The Green Hills Linker supports a comprehensive set of embedded features including automatic construction of C++ static constructor and destructor lists, incremental linking (-r), partial linking (-A), and full image maps in various formats.

Utility Programs

Green Hills provides 23 utility programs for format conversion, information retrieval, and other useful functions in support of the compilation process.

Optimizing C Compilers

The Green Hills Optimizing C Compilers are fully Unix compatible in command line options and C source language. The maximum compiler limits, such as number of functions per compilation, nesting level of loops, etc., meet or exceed those of both Unix C and ANSI C. Green Hills C has compiled the Unix kernel, libraries, and utilities for many different microprocessors. Green Hills C also supports the Berkeley Unix C extensions and Kanji in comments and string literals. Green Hills C conforms fully to ANSI X3.159-1989 Standard C (ISO/IEC 9899 and FIPS PUB 160).

C Versions

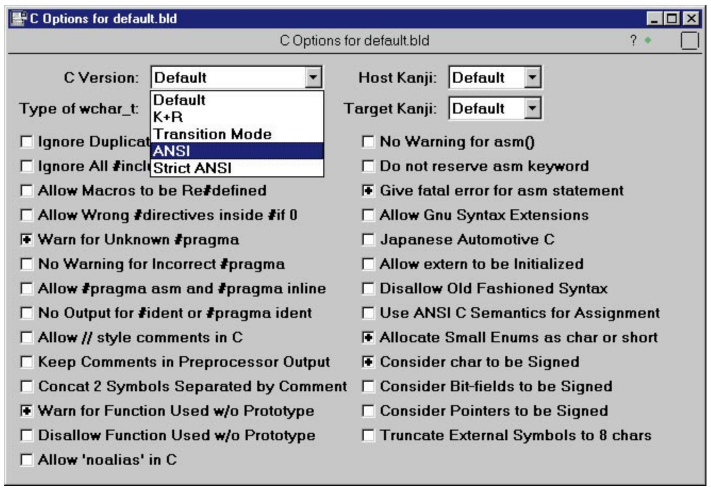
Green Hills C supports many versions of the C language:

- **K + R** - For C source files, interpret the source code as the C version documented in Kernigham & Ritchie, first edition, and compatible with the portable C compiler, or PCC.
- **Transition Mode** - Selects a mode of ANSI C compatibility similar to AT&T C. Issues 5.0 transition mode supporting function prototypes and the new ANSI keywords signed and volatile in a non-ANSI environment.
- **ANSI** - Sets the compiler in Permissive ANSI compatibility mode.
- **Strict ANSI** - Strict ANSI mode is 100% compliant with the ANSI X3.159-1989 standard and does not allow nonstandard constructs.

C Options

Green Hills C contains many specific language options so code can meet specific needs. These options include:

- **Ignore Duplicate #include** - Ignores an *#include* directive if attempting to include a file already included.
- **Allow macros to be Re#defined** - Suppresses the warning or error normally given when two *#define* directives provide different values for the same preprocessor symbol.
- **Allow #pragma asm and #pragma inline** - Allows the use of *#pragma asm*, *#pragma endasm*, and *#pragma inline* in C source files.
- **Wrong #directives inside #if 0** - During preprocessing, lines inside of false *#if*, *#elif*, *#ifdef*, and *#ifndef* are ignored.
- **No output for #ident or #pragma ident** - Prevents the compiler from outputting an ident directive in the assembly language output or from placing the same information in the .comment section when generating COFF or ELF object files directly.
- **Allow // style comments in C** - Allows C++ style comments, and C as well.
- **Concat 2 Symbols Separated by Comment** - Allows /* */ as concatenation in K & R C.
- **Allow GNU Syntax Extensions** - Supports GNU extensions, such as *#import*, zero size arrays, compound statements as part of expressions, inline functions, and the __inline__keyword.
- **Japanese Automotive C** - Enables a set of extensions to ANSI C used by Japanese automobile manufacturers.
- **Allow extern to be Initialized** - Allows variables declared with the extern storage class to accept initial values.
- **Disallow Old Fashioned Syntax** - Does not recognize outdated syntax for initializing variables, such as int i 5; and for assignment operators like =+, =-, =*.
- **Use ANSI C Semantics for Assignment** - Uses ANSI rules for ++ and *= in K&R C.
- **Allocate Small Enums as char or short** - Allocates enumerated types to the smallest storage possible.
- **Truncate External Symbols** - Truncate all symbol names to eight characters for compatibility with older compilers and linkers.



C Version: Default
Default
K+R
Transition Mode
ANSI
Strict ANSI

Type of wchar_t:

Host Kanji: Default

Target Kanji: Default

- ☐ Ignore Duplicat
- ☐ Ignore All #incl
- ☐ Allow Macros to be Re#defined
- ☐ Allow Wrong #directives inside #if 0
- ☑ Warn for Unknown #pragma
- ☐ No Warning for Incorrect #pragma
- ☐ Allow #pragma asm and #pragma inline
- ☐ No Output for #ident or #pragma ident
- ☐ Allow // style comments in C
- ☐ Keep Comments in Preprocessor Output
- ☐ Concat 2 Symbols Separated by Comment
- ☑ Warn for Function Used w/o Prototype
- ☐ Disallow Function Used w/o Prototype
- ☐ Allow 'noalias' in C
- ☐ No Warning for asm()
- ☐ Do not reserve asm keyword
- ☑ Give fatal error for asm statement
- ☐ Allow Gnu Syntax Extensions
- ☐ Japanese Automotive C
- ☐ Allow extern to be Initialized
- ☐ Disallow Old Fashioned Syntax
- ☐ Use ANSI C Semantics for Assignment
- ☑ Allocate Small Enums as char or short
- ☑ Consider char to be Signed
- ☐ Consider Bit-fields to be Signed
- ☐ Consider Pointers to be Signed
- ☐ Truncate External Symbols to 8 chars

C Build Options

Optimizing C++ Compilers

Green Hills C++ Optimizing Compiler conforms to the latest ANSI Standard for C++. Green Hills C++ is a "Scalable C++" compiler that includes a variety of user-selectable language feature combinations to meet specific needs.

Green Hills C++ compiler offers many options, including the new Embedded C++.

C++ Versions

Green Hills C++ supports many versions of the C++ language:

- **Standard (ANSI) C++ w/ warnings** - Enables ANSI C++ mode. Warning messages are issued when non-ANSI C++ features are used.
- **ARM** - Accepts the C++ language as defined by *The Annotated C++ Reference Manual (ARM)*, by Ellis and Stroustrop.
- **Embedded C++** - A subset of Standard C++.
- **Cfront 3.0** - Enables compatibility with cfront version 3.0.
- **Cfront 2.1** - Enables compatibility with cfront version 2.1.

C++ Options

Green Hills C++ contains many specific language options so code can meet specific needs. These options include:

- **Enable std namespace** - Enables implicit use of the standard namespace when standard header files are included.
- **Disable RTTI** - Disables support for runtime type information (RTTI) features "dynamic_cast" and "typeid".
- **Use long lifetimes for temps** - Creates temporary variables whose lifetime ends at the earliest end of scope, end of switch clause, or next label.
- **Keep comments in preprocessor output** - If producing a preprocessor output file passes comment lines through from the C++ source to the preprocessor output file.
- **Leave translated C** - Leaves a C version of the C++ code.
- **Enable exception handling** - Enables support for the C++ exception handling feature.
- **Disable array new/delete** - Enables support for the array new and delete features.

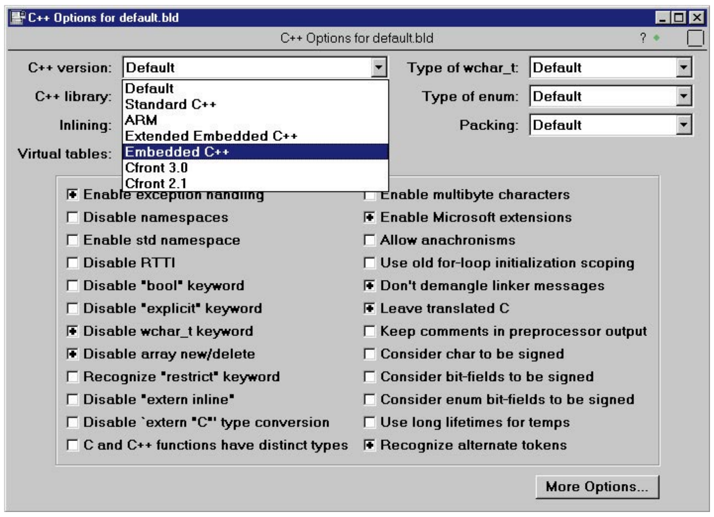
Embedded C++ (EC++)

Green Hills EC++ is a proper subset of ANSI C++ intended to meet the needs of embedded application developers. EC++ offers the same object-oriented benefits of C++, but with smaller code size, deterministic behavior, and a simpler user interface. EC++ hits the "sweet spot" between C and C++.

Many ANSI C++ features can result in added code size and performance overheads even if they are not used in an application. Simple expressions such as template specialization can generate code that is large or that takes a long time to execute. Library functions, particularly input-output, may call and include much more code than necessary for basic functionality. EC++ removes these limitations by omitting a number of C++ features that are not essential for most embedded applications because they increase code size and impair runtime efficiency. The features not defined in EC++ include:

- standard temple library
- templatized string, complex, or iostreams classes
- exception-handling functions or classes
- type_info class
- support for wide-character input/output, locals, or long double arithmetic

The Green Hills EC++ compilers can achieve improvements ranging from 30 -90 percent in code size and run-time efficiency over full ANSI C++, particularly for applications containing I/O.


C++ Options for default.bld

C++ version: Default
- Default
- Standard C++
- ARM
- Extended Embedded C++
- Embedded C++
- Cfront 3.0
- Cfront 2.1

Type of wchar_t: Default
Type of enum: Default
Packing: Default

[✓] Enable exception handling	[ ] Enable multibyte characters
[ ] Disable namespaces	[✓] Enable Microsoft extensions
[ ] Enable std namespace	[ ] Allow anachronisms
[ ] Disable RTTI	[ ] Use old for-loop initialization scoping
[ ] Disable "bool" keyword	[✓] Don't demangle linker messages
[ ] Disable "explicit" keyword	[✓] Leave translated C
[✓] Disable wchar_t keyword	[ ] Keep comments in preprocessor output
[✓] Disable array new/delete	[ ] Consider char to be signed
[ ] Recognize "restrict" keyword	[ ] Consider bit-fields to be signed
[ ] Disable "extern inline"	[ ] Consider enum bit-fields to be signed
[ ] Disable `extern "C"' type conversion	[ ] Use long lifetimes for temps
[ ] C and C++ functions have distinct types	[✓] Recognize alternate tokens

C++/EC++ Build Options

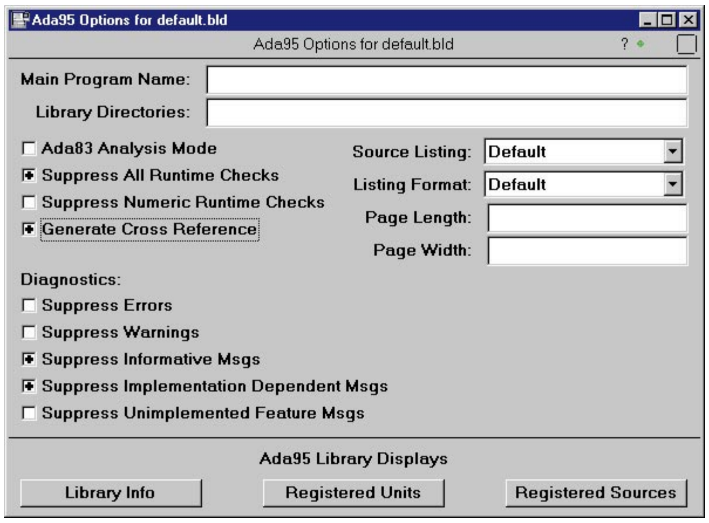
Optimizing Ada 95 Compilers

Green Hills ANSI Ada 95 are fully validated Ada 95 optimizing compilers for embedded targets running on Wind River System's VxWorks 5.3 under Tornado. Green Hills' embedded Ada 95 offers powerful new Ada language features and is available exclusively for Tornado applications.

Ada 95 Options

Green Hills Ada 95 contains specific language options so code can meet specific needs. These options include:

- **Ada83 Analysis Mode** - Gives useful hints on converting Ada83 code to Ada 95 code.
- **Suppress All Runtime Checks** - Suppresses all automatic run-time checking including numeric checking.
- **Suppress Numeric Runtime Checks** - Suppresses two kinds of Numeric Checks for the entire compilations: division_check and overflow_check.
- **Generate Cross Reference** - Generates a cross reference listing containing a line-numbered listing, followed by a cross reference table.
- **Diagnostics** - Informs the Builder in the AdaMULTI Development Environment what to display in the progress window when building an application.
- **Library Info** - Displays search path to libraries used by the application.
- **Registered Units** - Displays Unit Names and Unit descriptions of modules registered in the Ada library.
- **Registered Sources** - Displays source code path and names of modules registered in the Ada library.



Ada95 Options for default.bld

Main Program Name:
Library Directories:

☐ Ada83 Analysis Mode
☑ Suppress All Runtime Checks
☐ Suppress Numeric Runtime Checks
☑ Generate Cross Reference

Source Listing: Default
Listing Format: Default
Page Length:
Page Width:

Diagnostics:
☐ Suppress Errors
☐ Suppress Warnings
☑ Suppress Informative Msgs
☑ Suppress Implementation Dependent Msgs
☐ Suppress Unimplemented Feature Msgs

Ada95 Library Displays

Ada 95 Build Options

Ada 95 Features

Green Hills Ada 95 Optimizing Compilers are the first to successfully pass the latest suite of validation tests, ACVC 2.1, and achieve official certification for 16 host/target pairs. Green Hill's Ada 95 compiler implements all of the new enhancements defined in the ANSI/ISO/IEC-8652:1995 Ada 95 specification. These include:

- object-oriented programming
- hierarchical library organization
- type extensions of tagged types and child library units
- new task and synchronization features such as protected types

In addition, the compiler implements three optional Ada 95 annexes:

- **System Programming Annex (C)** - The Systems Programming Annex specifies additional capabilities provided for low-level programming. These capabilities are also required in many real-time, embedded, distributed, and information systems.
- **Real-Time Systems Annex (D)** - This Annex specifies additional characteristics of Ada implementations intended for real-time systems software. To conform to this Annex, an implementation shall also conform to the Systems Programming Annex.
- **Numerics Annex (G)** - The Numerics Annex addresses the particular needs of numerically intensive computing with regards to manipulation of complex numbers (computation and I/O), "strict" and "relax" mode of operation to cover both arithmetic and noncomplex elementary functions and random number generation, models of floating point and fixed point arithmetic applicable to "strict" mode, and various model attributes defined which are applicable to the "strict" mode for floating point types.

The compilers also provides specialized VxWorks and POSIX support that enables Ada 95 tasks to be implemented as either VxWorks tasks or POSIX threads (for self-hosted Unix applications).

OPTIMIZING FORTRAN COMPILERS

FORTRAN is the least standardized of all of the major languages. Computer system vendors have long exercised the right to add extensions to the basic language in order to provide themselves with a unique, proprietary product. Green Hills Optimizing FORTRAN 77 Compilers conform fully to ANSI X3.9-1978 Standard FORTRAN 77 (Full Language) and FIPS PUB 069-1, and adds the DOD MIL-STD 1753 FORTRAN extensions and selected VAX/VMS FORTRAN extensions.

FORTRAN VERSIONS

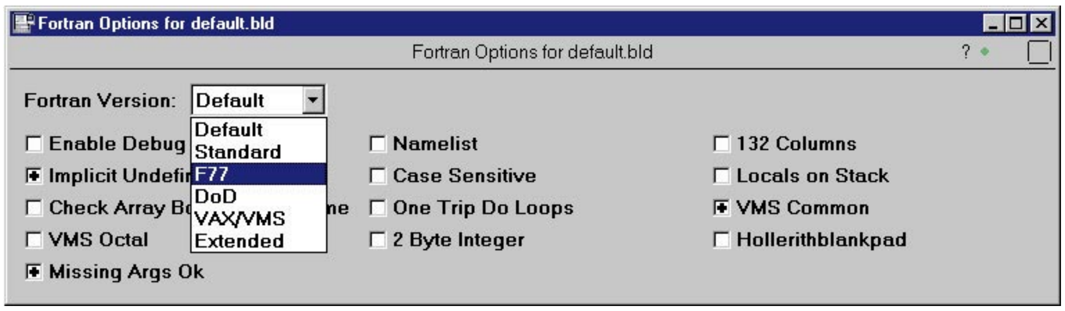
Green Hills FORTRAN contains specific language options so code can meet specific needs. These options include:

- **Standard** - Interprets FORTRAN code in compliance with the ANSI FORTRAN standard.
- **F77** - Interprets FORTRAN code for compatibility with AT&Ts f77 compilers.
- **DoD** - Enables DoD FORTRAN extensions.
- **Extended** - Allows as many general purpose language extensions as possible.

FORTRAN OPTIONS

Green Hills C contains many specific language options so code can meet specific needs. These options include:

- **Enable Debug Lines** - Compiles lines starting with d, D, x, or X.
- **Implicit Undefined** - The default data type for undeclared variables is "undefined', equivalent to coding IMPLICIT UNDEFINED(A-Z) at the top of the source file.
- **VMS Octal** - Controls whether a double quotation mark is used for octal characters.
- **Missing Args Ok** - Allows CALL X(1,,2). Suppresses warning resulting from the missing argument.
- **Namelist** - Enables the IBM and VMS compatible NAMELIST extensions in FORTRAN.
- **Case Sensitive** - Does not convert uppercase user-supplied variables to lowercase.
- **One Trip Do Loops** - Executes at least one iteration for every DO loop.
- **2 Byte Integer** - Sets the type for INTEGER to INTEGER*2.
- **132 Columns** - Extends source to interpret columns 1 through 132 instead of only 1 through 72.
- **Locals on Stack** - Allocates local variables to registers or stacks, equivalent to coding IMPLICIT AUTOMATIC (A-Z) at the start of every subroutine or function.
- **VMS Common** - Names COMMON blocks in the VMS style with a dollar sign appended.
- **Hollerithblankpad** - Pads hollerith constants on the right with blanks.



Fortran Options for default.bld

Fortran Options for default.bld

Fortran Version: Default

Default
Standard
F77
DoD
VAX/VMS
Extended

- [ ] Enable Debug
- [✓] Implicit Undefi
- [ ] Check Array Bo
- [ ] VMS Octal
- [✓] Missing Args Ok
- [ ] Namelist
- [ ] Case Sensitive
- [ ] One Trip Do Loops
- [ ] 2 Byte Integer
- [ ] 132 Columns
- [ ] Locals on Stack
- [✓] VMS Common
- [ ] Hollerithblankpad

FORTRAN Build Options

GREEN HILLS™ HIGH PERFORMANCE OPTIMIZING COMPILERS

C C++ EC++ Ada 95 FORTRAN

MULTI® INTEGRATED DEVELOPMENT ENVIRONMENT

Source Debugger	Run-Time Error Checking	Version Control	Program Builder
Profiler	Source and Class Browser	Editor	

PROCESSOR FAMILIES SUPPORTED

PowerPC	MIPS R3000/R4000/R5000	x86/Pentium	ARM	SH
680x0/683xx	MIPS 16	i960	TriCore	V810/V830/V850
ColdFire	RH32	RAD6000	Alpha	SPARC/SPARClite
M16	FR			

CROSS DEVELOPMENT TOOL CHAIN

Assemblers Linkers Librarians Object Code Converters Run-time Libraries

TARGET SYSTEMS SUPPORTED

Simulators ROM Monitors CPU Boards In-Circuit Emulators Processor Probes OCD/BDM/JTAG

REAL-TIME OPERATING SYSTEMS SUPPORTED

velOSity INTEGRITY Nucleus PLUS ThreadX VxWorks/Tornado ChorusOS pSOS+

HOST PLATFORMS FOR EMBEDDED DEVELOPMENT

Windows 95/NT SPARC/Solaris SPARC/SunOS PA-RISC/HP-UX Japanese Windows 95/NT

Green Hills SOFTWARE, INC.

Headquarters
30 West Sola Street
Santa Barbara, California 93101
Tel: 805.965.6044 • Fax: 805.965.6343
Email: sales@ghs.com • http://www.ghs.com

*MULTI is a registered trademark and Green Hills Software, velOSity, INTEGRITY, and the Green Hills Software logo are trademarks of Green Hills Software, Inc.
All other names mentioned are trademarks, registered trademarks, or service marks of their respective companies. Copyright © 1998 Green Hills Software, Inc. v1.0*