

# CodeWarrior® Development Tools



CodeWarrior® Development Studio is a complete integrated development environment (IDE) from hardware bring-up through programming embedded applications. By combining state-of-the-art debugging technology with the simplicity of a robust development environment, CodeWarrior Development Studio takes source-level debugging and embedded application development to a new level.

The CodeWarrior Development Studio provides a highly visual and automated framework that accelerates the development of even the most complex applications. Creating applications is fast and easy for developers of all experience levels. As a single development environment, it is consistent across all supported workstations and personal computers with an organization. On each of the supported platforms, the features and uses are identical. There is no need to worry about host-to-host incompatibilities. Learn once, use everywhere.

The CodeWarrior Development Studio contains all of the tools needed to complete a major embedded development project.

#### **Complete Set of Tools**

From text editors to compilers and debuggers, CodeWarrior development studio provides

everything the professional embedded developer needs:

- CodeWarrior C/C++ Compiler Suite
- Runtime libraries
- Assembler
- Standard template library (STL)

#### **Source-Level Debugger**

Provides a high-performance windowed source-level debugger equipped with the latest productivity—enhancing graphical features to shorten board bring-up and application development time; uses the symbolics database to provide source-level debugging; supports symbol formats such as Microsoft® CodeView®, Debug With Arbitrary Records Format (DWARF) and STABS.

**Instruction Set Simulator**

Jump-starts application development with an integrated instruction set simulator (ISS).

**Project Manager**

Handles top-level file management for the software developer; organizes project items by major group, such as files and targets; tracks state information (such as file modification dates); determines build order and inclusion of specific files in each build; coordinates with plug-ins to provide services like version-control and RTOS support.

**Text Editor**

Enables the creation and manipulation of source code and other text files and is completely integrated with other IDE functions.

**Search Engine**

Finds a specific text string or replaces found text with substitute text across multiple files and directories; allows use of regular expressions; provides file-comparison and differencing functionality.

**Source Browser**

Maintains a symbolics database for the program (examples of symbols include names and values of variables and functions); uses the symbolics database to assist code navigation; links every symbol to other locations in the code related to that symbol; processes both object-oriented and procedural languages.

**Debugger**

By combining a state-of-the-art IDE with the simplicity of a windowed environment, the debugger takes C/C++ source-level debugging to a new level. The debugger assembles a wide array of high-powered components and features into a powerful GUI to help get projects completed and to market ahead of schedule and under budget.

All of the debugger hardware and software features provide simple access and execution. Any debug operation desired is done through an intuitive “point-and-click” interface to make debugging fast, flexible and easy.

**Window-Based Workspace Environment**

The debugger enables developers to operate more efficiently with user-friendly debugging, multiple windows, point-and-click capabilities and outline format.

The debugger’s interface allows users to customize the workspace to fit their needs: to create custom buttons, toolbars and menus, and to “float” windows that are an integral part of the debugger so that they become independent windows on the work station. This provides increased visibility and control over the display of information in the debugger. Windows that have been separated from the debugger can also be “docked” to rejoin the main debugger workspace controls.

The debugger’s workspace allows users to focus on complex debugging tasks. Each workspace contains just the set of views needed for the task at hand. The application workspace provides a high-level view of the target software, while the hardware workspace provides a low-level view of the target hardware.

**Seamless Integration**

The debugger is fully integrated with a variety of run-control devices like Ethernet TAP and CodeWarrior USB TAP, resulting in optimized run-control and faster downloads.

**Full-Featured Debugging**

The debugger provides a rich set of debugging features designed to help the developer quickly find and repair software defects.

**Stack Crawl Window**

Displays information about a suspended thread or process while debugging—all in the primary window used during a debug session. The stack crawl window can be used to view the call stack for a function, view function variables and global data, view a routine in source, assembler or mixed-mode, view the current program counter indicating the statement about to be executed, view and set breakpoints for the current function or single-step through the current function from the program counter.

**Single-Stepping**

Supports the common single-stepping mechanisms (Step Into, Step Over, Step Out), stepping statements in source-level display and instructions in Assembler and mixed-mode level display. Single-stepping can be performed in any of the source code display windows (stack crawl window, class browser window, source file window and symbolics window).

**Contextual Data**

Displays data values at a glance for variables in source windows during execution. In addition to standard toolbar tool tips common to applications, the debugger reveals contextual data when developers mouse over the variable in the source statement.

**Preprocessor Information**

Provides clear understanding of the code being generated and relevant scoping used during the build cycle in the current debug context.

## Breakpoints

Sets breakpoints in source display windows with a simple click on the window's left margin or by dragging a source statement or assembly instruction to the breakpoints window. A separate breakpoints window in the debugger allows developers to view all breakpoints consolidated into a single worksheet and save them to disk for use later or for sharing with other project members.

## CodeWarrior Debugger Offers Many Different Types of Breakpoints

### Eventpoints

Eventpoints are used to perform a task when program execution arrives at a specific line of source code or when an associated conditional expression evaluates to true. Developers can set an eventpoint that performs a task such as logging or speaking a string or expression and then recording messages to the log window, pausing execution just long enough to refresh debugger data, running a script, playing a sound, skipping execution of a line of source code or collecting trace data. An eventpoint is equivalent to a breakpoint that performs a task other than halting program execution.

### Watchpoints

Watchpoints halt program execution when a specific location in memory changes value. After a developer sets a watchpoint at a key point in memory, he/she can halt program execution when that point in memory changes value or, for some devices, when the memory location is accessed, examine the call chain, check register and variable values, and step through the code. Developers can also change values and alter the flow of normal program execution. A watchpoint is equivalent to a memory breakpoint.

## Special Breakpoints

Special breakpoints halt program execution for very specific reasons, such as when program execution arrives at the beginning of the function `main()`, a C++ exception occurs or an event occurs that the debugger plug-in has defined as a break event. Developers cannot change or delete special breakpoints, but they can enable and disable them.

## Memory Window

Displays data in various data formats: raw hexadecimal and ASCII, standard C data types, structures and enumeration types from the current debug project, or as machine instructions either in machine language assembly or displayed related to source code. The memory window displays data in a "raw" hexadecimal format, as well as ASCII, with the word-length variable through a drop-down selection at the bottom of the screen. The memory window allows for the display and editing of memory locations. Allows for editing in the context of the current display format—for example, "view as int" allows editing an integer value. Establishes selected memory addresses as a watchpoint for conditional execution simply by right-click mouse selection.

## Register Window

Shows a complete list of available processor and peripheral registers, both built-in and memory mapped. Registers are grouped by function in a hierarchical view that lets developers browse in a single window or open a new window for a specific group. Registers are shown for multiple processes, threads or processors. Register values are shown and can be modified in a variety of formats (signed decimal, hexadecimal, etc.). The display of register groupings and memory mapped registers is controlled by a user-modifiable XML register description file.

## Register Details Window

Shows the contents of a given register in a format similar to a processor data book. Register and bit field descriptions are included along with logical groupings of bits into value fields. Bit field values can be modified manually or from a list of relevant value choices. The display of register details is controlled by a user-modifiable XML register description file.

## Cache Window

Displays current values and state of cache contents for each data and instruction cache within the target processor. Cache writes allow developers the freedom to edit cache contents and experiment with cache behavior as it relates to the current code being debugged.

## Object File Format

Supports STABS and ELF/DWARF (version 1 and 2) object file output formats.

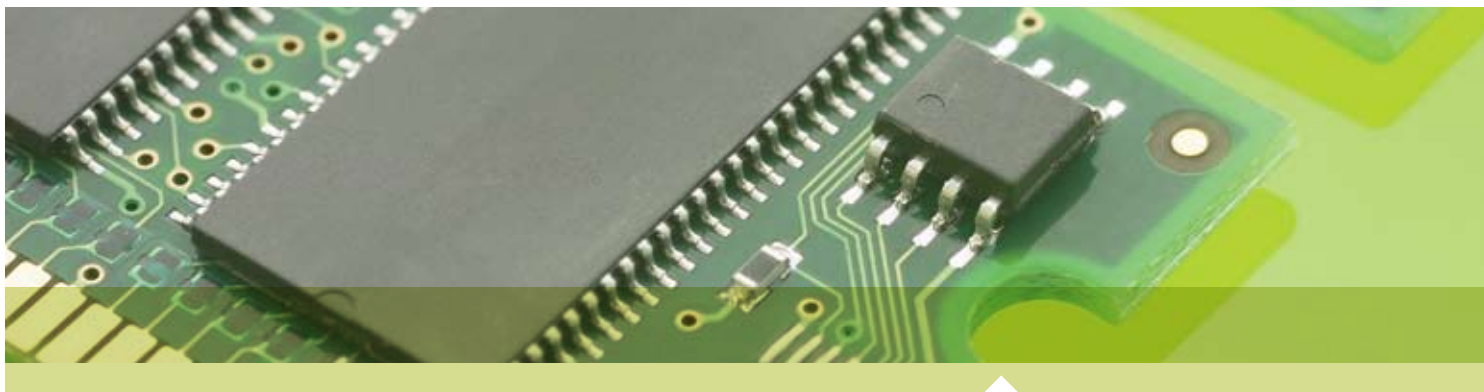
## Multi-Thread/Process Debugging\*

Enables debugging of multi-threaded applications and multiple processes. Debugging can select and debug a single thread, multiple threads or all threads—based on debug preferences. Each thread being debugged can be managed through the threads window or separate stack crawl windows per thread.

*\*This capability is not available for all processors.*

## Multi-Core Debugging

Complete quad-core debugging functionality integrated in one development environment. With a single instance of the CodeWarrior debugger running, developers are capable of debugging a four-processor SoC through four separate process stack crawl windows and subsequently derived windows for that processor. As a developer switches context between the stack crawl windows, appropriate windows and menu items for processor cache are adjusted.

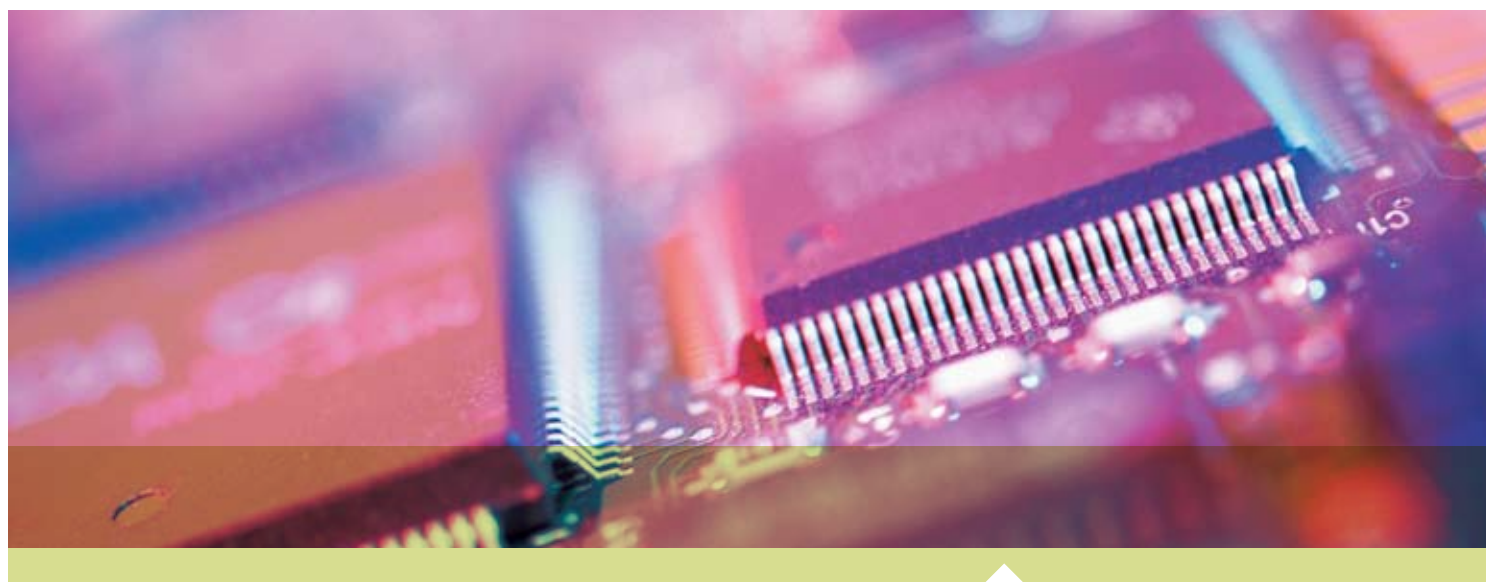


### Multiple CPU Debugging

Debugs up to 255 processors in a symmetric multiprocessor array—each running multiple processes and threads. For those computers with SMP capabilities, the processors window is used to define which processor specifically will be placed into stop-mode debugging.

### Target Connection Wizard

Simplifies and automates the task of defining new connection definitions based on hardware and communication parameters.



### Other CodeWarrior Debugger Tools

#### Board Bring-Up\*

Helps developers deal with the complexity of bringing up a board by providing complete control over all board settings, including initial register values and memory configuration. After initial target register values are defined, the debugger restores these values each time the user connects to the board. Then an assembler source file can be created from these settings as an addition to the project. The debugger also includes a comprehensive set of hardware diagnostics and robust flash programming to support an extensive list of flash devices.

*\*This capability is only available with 32-bit processors.*

#### Flash Programming

Programs on-board flash devices from within the same GUI used to troubleshoot the application. No boot code is required to run on the target system in order to use the programming features of the CodeWarrior flash programmer.

### Command-Line Window

Developers can use the command-line interface together with various scripting engines such as the Microsoft Visual Basic® script engine, the JavaScript™ engine, Tcl, Python and Perl to automate test/SW/HW validation. Developers can also issue a command line that saves a log file of command-line activity.

#### Logic Analyzer

Developers can utilize the debugger with the logic analyzer to troubleshoot low-level hardware components in order to understand complex signals on an embedded hardware platform.

Freescale has implemented an interface to seamlessly integrate logic analyzer communications into the debugger.

Features include:

- Trace on/off
- Trace everything
- Trace history
- Start trace based on specified address
- Start trace on address range
- Trace all in address range
- Breakpoint on trigger
- Trigger tracing on breakpoint
- Support for Tektronix® and Agilent test equipment

#### Hardware Diagnostics\*

The CodeWarrior Development Studio comes with diagnostics that enable the developer to determine if the basic hardware is functional.

These tests include:

- Memory Read/Write—Performs diagnostic tests for performing memory reads and writes over the remote connection interface.
- Scope Loop—Configures diagnostic tests for performing repeated memory reads and writes over the remote connection interface. The tests repeat until the developer stops them. By performing repeated read and write operations, developers can use a scope analyzer or logic analyzer to debug the hardware device.
- Memory Tests—Lets developers perform three different tests on the hardware: Walking Ones, Address or Bus Noise.

*\*This capability is not available for all products.*

Developers can specify any combination of the tests and the number of passes to perform them. For each pass, the hardware diagnostic tools perform the tests in turn, until all passes are complete. The tools tally memory test failures and display them in a log window after all passes are complete. Errors resulting from memory test failures do not stop the testing process; however, fatal errors immediately stop the testing process.

#### **CodeWarrior Instruction Set Simulator\***

Provides a quick and easy way to begin developing code without the requirement for access to hardware. The ability to develop software without requiring hardware provides a number of significant benefits to software engineers, including the ability to run code before custom hardware is available, running/testing code when hardware resources are limited, and learning how to use the development environment without first having to get hardware running.

The CodeWarrior ISS provides full instruction simulation and supports standard C library I/O. It is fully integrated with the CodeWarrior IDE and also provides a full command-line interface. The CodeWarrior ISS is available for specific platforms.

*\*Simulator varies by processor family.*

## **Compiler**

CodeWarrior Development Studio combines industry-leading components to offer embedded developers all the necessary tools to create, build and deploy quality products to their customers. One major component of the IDE is the CodeWarrior compiler. It combines industry-proven optimization technology with the versatility and control needed to fully exploit today's complex PC CPUs. CodeWarrior Compiler's design is based on a partitioned architecture that results in proven reliability and flexibility for embedded applications, as well as interoperability with other CodeWarrior development products.

The compiler provides language-specific front ends for C, embedded C++ and C++ that parse the original source code into a common token-based representation of the source. Optimizations are applied to this intermediate language representation. Also, the fully optimized code is converted into the appropriate machine code via a robust, table-driven back-end module. Freescale's integration of silicon design with our compiler team, combined with the compiler's modular design, make it possible for the CodeWarrior portfolio to provide highly optimized compilers for new silicon with very short lead times. The compiler's modular architecture enables developers to immediately gain maximum performance from their compiler/silicon investments.



## Proven Optimization Technology

The CodeWarrior compiler produces exceptionally fast, compact, high-quality object code. A large number of highly refined, global, local, CPU-specific and sometime application-specific (profile-driven) optimization techniques enable the programmer to fine-tune the compiler's output to match the application's requirements. Programmers can select various optimizations to balance execution speed with code size while intelligent defaults can generate optimal code out of the box.

## Full-Featured Compiler

The compiler provides a rich set of features and components:

### Advanced C/Embedded C++/C++ Compiler

Designed for highly embedded development support.

Key features include:

- Advanced optimization technology to generate fast, compact, high-quality code
- Field-proven reliability to meet extreme embedded design constraints
- Compatibility with ANSI C++ specs (ISO/IEC 14882:1998E) and ANSI C specs (X3.159-1989 and most compilers are also ANSIC99 [except complex numbers])
- Standards conformance (ANSI and EABI) for maximum tool interoperability
- Complete control of code and data memory allocation
- Options to pack or byte-swap structures to match existing data types
- Support for position-independent code (PIC) and data (PID)
- Board support routines for bare-board applications (no OS)
- Proven performance with industry-leading RTOSs

## Assembler

Full-featured macro assembler that is automatically invoked by the project manager or as a complete stand-alone assembler for generating object modules. Key features include:

- Conditional macro assembler with over 30 directives
- Unlimited number of symbols
- Debug information for source-level debugging of assembly programs

## Linker

Offers precise control over the allocation, placement and alignment of code and data in memory. Key features include:

- Links object modules into absolute or relocatable modules
- Reads/writes/mixes ELF and STABS object files
- Generates fully EABI-compliant ELF/DWARF 2.0 output for consumer tool interoperability

## Libraries

The standard libraries include:

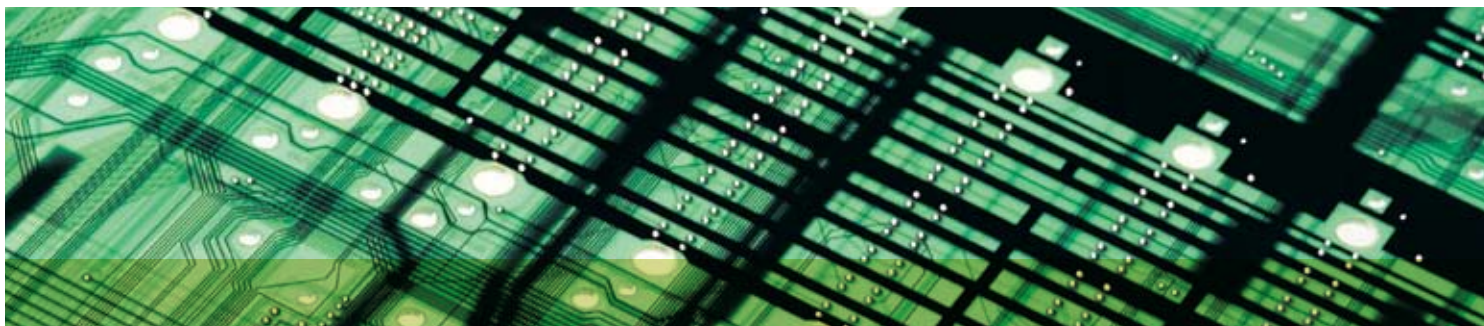
- Complete C++ library (STL)
- Complete, reentrant C libraries compliant with ANSI/ISO, POSIX and SVID standards
- Multithreading
- Full complement of math libraries, including IEEE-754 appendix functions
- Efficient floating-point libraries for fast execution of calculations

## Profiler

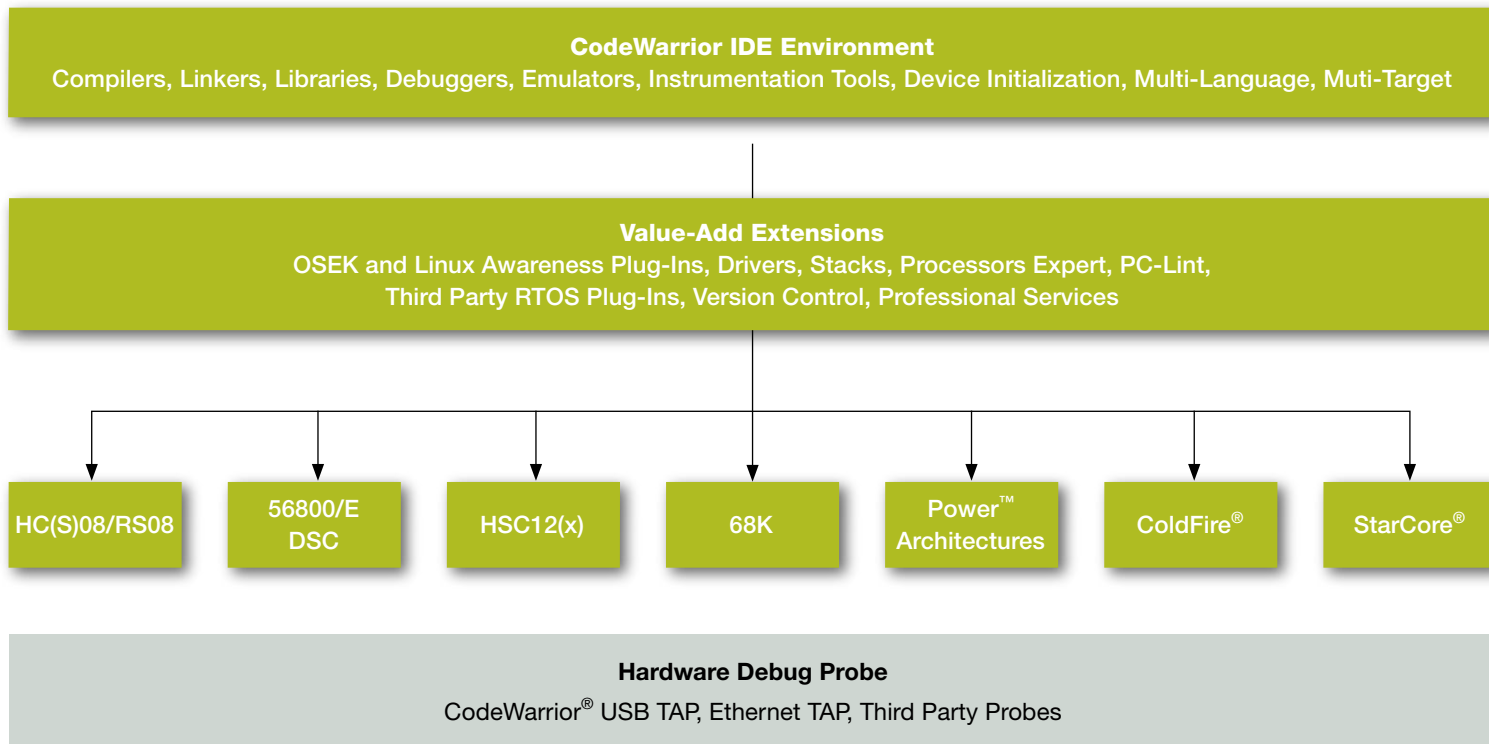
Profiling options contained in the compiler instrument application code, which when executed save profile information that can be viewed by the profiler utility. This profile data can also be used by the compiler for additional code optimization based on execution paths.

## Documentation

The IDE and compiler ship with extensive documentation specific to the chosen architecture. The Getting Started Guide enables developers to quickly get up-to-speed and enhances out-of-box experience. In addition to hardcopy, all manuals are available in HTML and PDF formats online.



## CodeWarrior Product Diagram



## Project Manager

The CodeWarrior Development Studio Project Manager provides a powerful framework to simplify organizing, configuring and building complex development projects and automating many aspects of managing a project.

The project manager performs automatic dependency analysis and generates the appropriate project context. The powerful graphical user interface (GUI) enables the user to configure a project by selecting from menus, the options covering everything from optimization level, debugging level and language-specific features to target type (executable or library). It takes the developer step-by-step through a series of questions to create a project and includes example stationery (a template) as a starting place for the application. The stationery includes a linker command file and project files that make it possible to associate debug connections easily and is provided for every CPU and programming language supported by the CodeWarrior compiler.

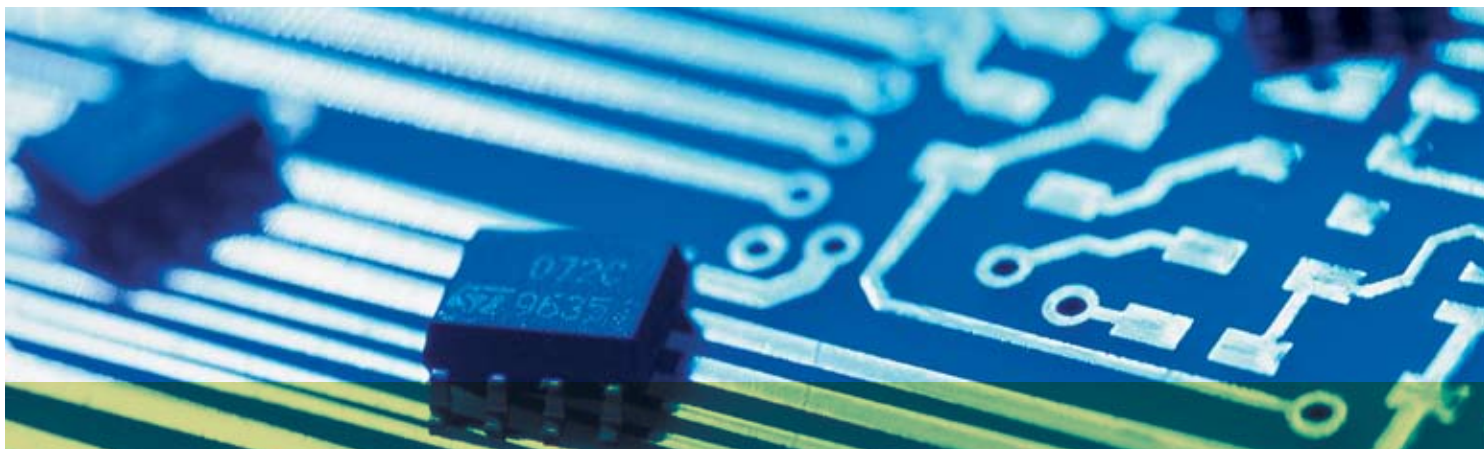
## Text Editor

CodeWarrior Development Studio includes a full-featured, user-configurable, windowed text editor with features such as syntax coloring and auto-indenting. Syntax coloring helps quickly identify language keywords and constructs, including comments, strings, constants and more. The CodeWarrior text editor implements all of the standard functions that are expected from an editor, including a powerful search feature that can find values within multiple files. It is fully configurable, so the developer can change the key bindings, font type, font size, color scheme, syntax coloring and provides a single, consistent editor interface for all host and target development combinations. The text editor is an integral part of the overall CodeWarrior Development Studio and can be invoked and controlled as an object from other components within the CodeWarrior Development Studio.

## Search Engine

Industry observers estimate that software developers spend nearly half their time searching for basic information buried in application code. As applications grow in complexity, the time required to find, analyze and modify code grows as a proportion of total engineering effort. The search engine reduces this largely unproductive time by integrating code browsing and searching into a single tool. It is fast and provides emantic code navigation to make it possible to find specific code structures, symbols or pattern among hundreds of directories.

The seamless integration between the search engine and the text editor means all changes in the code are immediately reflected in the browser without any recompilation. With the search engine, the mouse can be used to navigate between the different symbols by placing the mouse cursor on a symbol and right-click to invoke the text editor. This will open the file and highlight the exact location of the selected symbol.



Product Name	Supports Freescale Device Family
<b>Software Development Tools</b>	
CodeWarrior® Development Studio for Microcontrollers	RS08, HC(S)08, ColdFire® V1
CodeWarrior Development Studio for HCS12(X) Microcontrollers	HCS12 or HCS12X/XGATE
CodeWarrior Development Studio for ColdFire Architectures	ColdFire Architectures (V2, V3, V4)
CodeWarrior Development Studio for Power Architecture™ Processors	Power Architecture Technologies for Networking Applications
CodeWarrior Development Studio for mobileGT	Power Architecture Technologies for Information and Multimedia Applications
CodeWarrior Development Studio for MPC55xx	Power Architecture Technologies for Telematics Applications
CodeWarrior Development Studio for MPC5xx	MPC5xx Processors
CodeWarrior Development Studio for StarCore® DSP	MSC81xx Processors
CodeWarrior Development Studio for StarCore and SDMA	i.300, MXC300, MXC275 Multi-Core Architectures
CodeWarrior Development Studio for 56800/E Digital Signal Controllers	56F80x/2x, 56F85x, 56F81xx, 56F83xx, 56F801x, 56F802x/3x Digital Signal Controllers
CodeWarrior Development Studio for 68K	<b>68K Embedded Systems</b>
<b>Value-Added Extensions</b>	
OSEK <i>turbo</i>	OSEK/VDX™ Compatible
Processor Expert™	Integration, Configuration and LLD
Version Control	Integrated VCS for Check-In/Check-Out
<b>Hardware Debug Probe</b>	
CodeWarrior Ethernet TAP	
CodeWarrior USB TAP	

**Learn More:** For current information about Freescale products and documentation, please visit [www.freescale.com/codewarrior](http://www.freescale.com/codewarrior).