

C++test™ for TI Code Composer Development Environment

DATA SHEET

Parasoft C++test – Comprehensive Code Quality Tools for TI Code Composer Development Environment

Parasoft C++test is an integrated code quality tool suite that helps developers write better code and test it more efficiently. C++test provides teams a practical way to ensure that their C and C++ code works as expected by coding policy enforcement and early error detection.

Critical time-proven best practices, such as **static analysis**, **comprehensive code review**, **memory analysis**, and **unit and component testing** with integrated **coverage analysis** are enabled in the developers' desktop, early in the development cycle. Integration with Parasoft Concerto, project management and reporting system, facilitates visibility into project status and trends based on C++test results and other key process metrics.

C++test Code Composer Integration

The C++test plugin for TI Code Composer provides CCS users easy access to C++test's **full static code analysis**, **code review**, **runtime memory analysis** and **unit testing** capabilities directly within their IDE. This seamless integration enables testing and verification to become a natural and continuous part of the development process. The complete target-based test execution flow, including test case generation, cross-compilation, deployment, execution and loading results back to the GUI, can be automated within C++test.

C++test is available as a plugin to TI Code Composer v. **4.x**, as well as an Eclipse-based standalone version that supports Code Composer **3.x** projects import.

C++test supports the following compilers:

TMS320C6x C/C++ Compiler v5.1, v6.0, v6.1; TMS320C2000 C/C++ v4.1, v5.2; MSP430 C/C++ Compiler v3.2

Automate Code Analysis for Monitoring Compliance

A properly implemented coding policy can eliminate entire classes of programming errors by establishing preventive coding conventions. C++test statically analyzes code to check compliance with such a policy. Hundreds of built-in rules including implementations of MISRA, MISRA 2004, and the new MISRA C++ standards, as well as guidelines from Meyers' Effective C++ and other popular sources help identify potential bugs from improper C/C++ language usage, enforce best coding practices, and improve code maintainability and reusability. Custom rules, which are created with a graphical RuleWizard editor, can enforce standard API usage and prevent the recurrence of application-specific defects after a single instance has been found.

Monitor the Application for Memory Problems

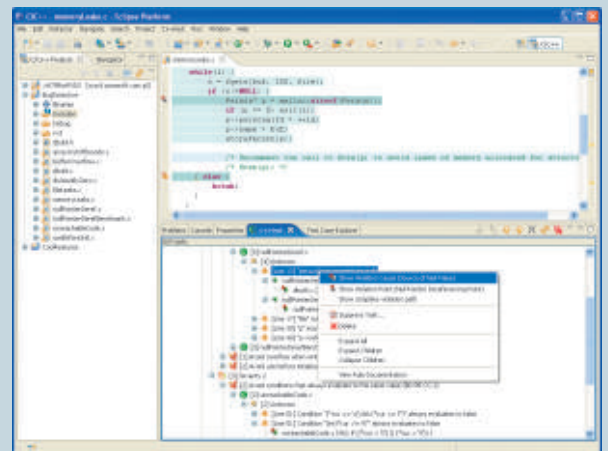
Application memory monitoring is the best known approach to eliminating serious memory-related bugs with zero false positives. The running application is constantly monitored for certain classes of problems—like memory leaks, null pointers, uninitialized memory, and buffer overflows—and results are visible immediately after the testing session is finished.

Without requiring advanced and time-consuming testing activities, the prepared application goes through the standard functional testing and all existing problems are flagged. The application can be executed on the target device, simulated target, or host machine. The collected problems are presented directly in the developer's IDE with the details required to understand and fix the problem (including memory block size, array index, allocation/deallocation stack trace etc.)

Coverage metrics are collected during application execution. These can be used to see what part of the application was tested and to fine tune the set of regression unit tests (complementary to functional testing).

Application Memory Monitoring

- Identify complex memory-related problems through simple functional testing—for example:
 - memory leaks
 - null pointers
 - uninitialized memory
 - buffers overflows
- Collect code coverage from application runs
- Increase the testing results accuracy through execution of the monitored application in a real target environment



C++test's BugDetective identifies critical bugs without executing the code

Identify Runtime Defects Without Executing Software

The advanced interprocedural static analysis module of C++test simulates feasible application execution paths, which may cross multiple functions and files, and determines whether these paths could trigger specific categories of runtime bugs. Defects detected include using uninitialized or invalid memory, null pointer dereferencing, array and buffer overflows, division by zero, memory and resource leaks, and various flavors of dead code. C++test greatly simplifies defect analysis by providing a complete analyzed path trace for each potential defect in the Eclipse IDE. Automatic cross-links to code help users quickly jump to any point in the highlighted analysis path.

Automated Code Review

Code review is known to be the most effective approach to uncover code defects. The C++test Code Review module automates preparation, notification, and tracking of peer code reviews. Status of all code reviews, including all comments by reviewers, is maintained and automatically distributed by the C++test infrastructure. C++test supports two typical code review flows using the IDE facilities:

- Post-commit code review. This mode is based on automatic identification of code changes in a source repository via custom source control interfaces, and creating code review tasks based on pre-set mapping of changed code to reviewers.
- Pre-commit code review. Users can initiate a code review from the desktop by selecting a set of files to distribute for the review, or automatically identify all locally changed source code.

Unit and Integration Test with Coverage Analysis

C++test's automated testing helps establish correctness and reliability of newly-developed or legacy code. C++test automatically generates complete tests, including test drivers and test cases for individual functions, purely in C or C++ code in a format similar to CppUnit. Alternatively, users can interactively define tests using a Test Wizard. These tests, with or without modifications, are used for initial validation of the functional behavior of the code. By using corner case conditions, these automatically generated test cases also check function responses to unexpected inputs, exposing potential reliability problems. To expand the range of test conditions without increasing test code, tests can be parameterized using data sources.

A multi-metric test coverage analyzer, including statement, branch, path, and MC/DC coverage, helps users gauge test suite efficacy and completeness, and demonstrate compliance with test and validation requirements, such as DO-178B. Coverage reports including file, class, and function data can be produced in a variety of formats.

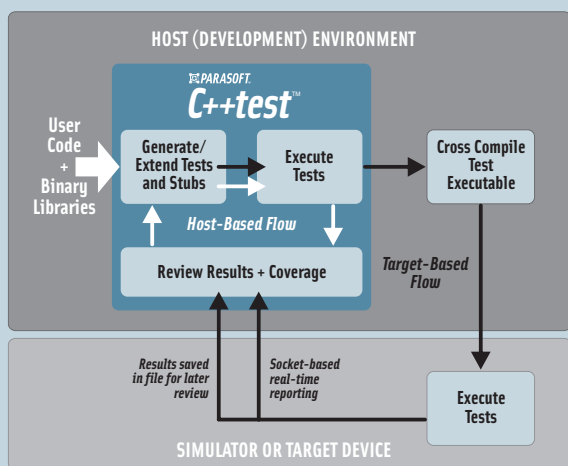
Test on the Host, Simulator, and Target

C++test supports testing on both host or simulator and target hardware. C++test automates the complete test execution flow, including test case generation, cross-compilation, deployment, execution, and loading results (including coverage metrics) back into the GUI. Testing can be driven interactively from the GUI or from the command line for automated test execution, or performed through batch regression testing. In the interactive mode, users can run tests individually or in selected groups for easy debugging or validation. For batch execution, tests can be grouped based either on the user code they are linked with, or their name or location on disk.

High Degree of Customization

C++test allows full customization of its test execution sequence. In addition to using the built-in test automation, users can incorporate custom test scripts and shell commands to fit the tool into their specific build and test environment. This unparalleled flexibility enables users to realize their desired test flow without being constrained by the preset tool options.

C++test can be utilized with a wide variety of embedded OS and architectures, by cross-compiling the provided runtime library for a desired target runtime environment. All test artifacts of C++test are source code, and therefore completely portable.



C++test's customizable workflow allows users to test code as it's developed, then use the same tests to validate functionality/reliability in target environments

Advanced Unit Test features:

- Automatic generation of tests and stubs
- Automatic generation of assertions based on observed test results
- Graphical Test Case Wizard for interactive definition of tests
- Complete visibility into test and stub source code
- Intelligent, test-case-sensitive stubs
- Parameterization of tests and stubs
- Multi-metric coverage analysis (including MC/DC)
- Flexible support for continuous regression testing
- Annotation of tests against bug and requirement IDs
- Execution of tests under debugger
- Special mode for testing template code

www.parasoft-embedded.com

Contact info:

Parasoft Corporation, 101 E. Huntington Dr., 2nd Flr., Monrovia, CA 91016

Ph: (888)305.0041, Fax: (626)256.6884, Email: info@parasoft.com