PARASOFT®

C++test™

C/C++ Development

DATA SHEET

Parasoft® C++test™ – Comprehensive Code Quality Tools for C/C++ Development

Parasoft C++test enables teams to produce better code, test it more efficiently, and consistently monitor progress towards their quality goals. With C++test, critical time-proven best practices—such as static analysis, comprehensive code review, runtime error detection, unit and component testing with integrated coverage analysis—are automated on the developer's desktop, early in the development cycle. A command line interface enables fully automated execution within regression and continuous integration environments, providing data for monitoring and analyzing quality trends. Moreover, C++test integrates with Parasoft's Concerto, which provides interactive Web-based dashboards with drill-down capability. This allows teams to track project status and trends based on C++test results and other key process metrics.

For embedded and cross-platform development, C++test can be used in both host-based and target-based code analysis and test flows. See page 3 for details.

### 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. To configure C++test to enforce a coding standards policy specific to their group or organization, users can define their own rule sets with built-in and custom rules. Code analysis reports can be generated in a variety of formats, including HTML and PDF.

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 Effective STL books, 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.

Identify Runtime Bugs without Executing Software

BugDetective, 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 path trace for each potential defect in the developer's IDE. Automatic cross-links to code help users quickly jump to any point in the highlighted analysis path.
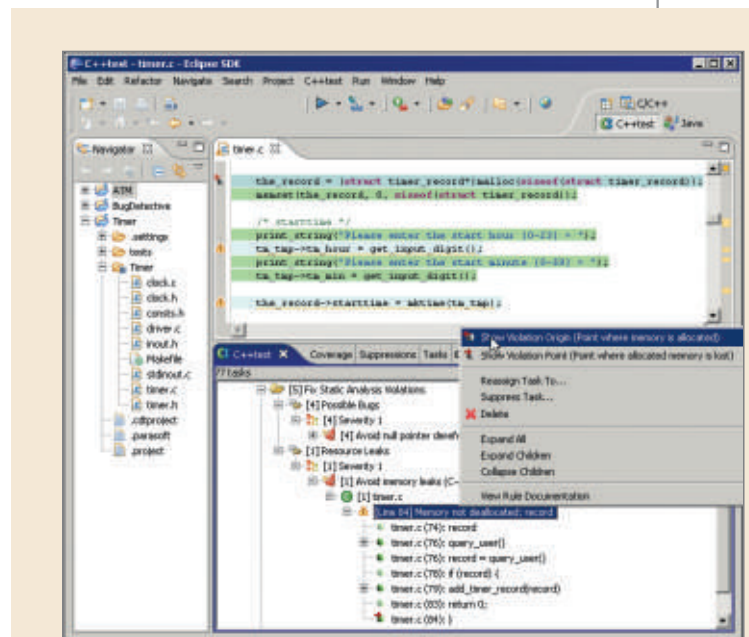
C++test's static analysis identifies critical bugs without executing the code (Eclipse version shown)

Benefits

- **Increase team development productivity** — Apply a comprehensive set of best practices that reduce testing time, testing effort, and the number of defects that reach QA.
- **Achieve more with existing development resources** — Automatically vet known coding issues so more time can be dedicated to tasks that require human intelligence.
- **Build on the code base with confidence** — Efficiently construct, continuously execute, and maintain a comprehensive regression test suite that detects whether updates break existing functionality.
- **Gain instant visibility into C and C++ code quality and readiness** — Access on-demand objective code assessments and track progress towards quality and schedule targets.
- **Reduce support costs** — Automate negative testing on a broad range of potential user paths to uncover problems that might otherwise surface only in "real-world" usage.

Features

- Static analysis of code for compliance with user-selected coding standards
- Graphical RuleWizard editor for creating custom coding rules
- Static code path simulation for identifying potential runtime errors
- Automated code review with a graphical interface and progress tracking
- Application monitoring/memory analysis
- Automated generation and execution of unit and component-level tests
- Flexible stub framework
- Full support for regression testing
- Code coverage analysis with code highlighting
- Code coverage analysis for unit testing and beyond (including application-level tests)
- Full team deployment infrastructure for desktop and command line usage

Runtime Error Detection

- 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 test result accuracy through execution of the monitored application in a real target environment

Streamline Code Review

Code review is known to be the most effective approach to uncover code defects. Unfortunately, many organizations underutilize code review because of the extensive effort it is thought to require. The C++test Code Review module automates preparation, notification, and tracking of peer code reviews, enabling a very efficient team-oriented process. 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:

- **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.

The effectiveness of team code reviews is further enhanced through C++test's static analysis capability. The need for line-by-line inspections is virtually eliminated because the team's coding policy is monitored automatically. By the time code is submitted for review, violations have already been identified and cleaned. Reviews can then focus on examining algorithms, reviewing design, and searching for subtle errors that automatic tools cannot detect.

Monitor the Application for Memory Problems

Runtime Error Detection 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).

Unit and Integration Test with Coverage Analysis

C++test's automation greatly increases the efficiency of testing the 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. 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.

Test creation and management is simplified via a set of specific GUI widgets. A graphical Test Case Wizard enables developers to rapidly create black-box functional tests for selected functions without having to worry about their inner workings or embedded data dependencies. A Data Source Wizard helps parameterize test cases and stubs—enabling increased test scope and coverage with minimal effort. Stub analysis and generation is facilitated by the Stub View, which presents all functions used in the code and allows users to create stubs for any functions not available in the test scope—or to alter existing functions for specific test purposes. Test execution and analysis are centralized in the Test Case Explorer, which consolidates all existing project tests and provides a clear pass/fail status. These capabilities are especially helpful for supporting automated continuous integration and testing as well as "test as you go" development.

A multi-metric test coverage analyzer, including statement, branch, path, and MC/DC coverage, helps users gauge the efficacy and completeness of the tests, as well as demonstrate compliance with test and validation requirements, such as DO-178B. Test coverage is presented via code highlighting for all supported coverage metrics—in the GUI or color-coded code listing reports. Summary coverage reports including file, class, and function data can be produced in a variety of formats.

Automated Regression Testing

C++test facilitates the development of a robust regression test suite that detects if incremental code changes break existing functionality. Whether teams have a large legacy code base, a small piece of just-completed code, or something in between, C++test can generate tests that capture the existing software behavior via test assertions produced by automatically recording the runtime test results. As the code base evolves, C++test reruns these tests and compares the current results with those from the originally captured "golden set." It can easily be configured to use different execution settings, test cases, and stubs to support testing in different contexts (e.g., different continuous integration phases, testing incomplete systems, or testing specific parts of complete systems). This type of regression testing is especially critical for supporting agile development and short release cycles, and ensures the continued functionality of constantly evolving and difficult-to-test applications.

Configurable Detailed Reporting

C++test's HTML, PDF, and custom format reports can be configured via GUI controls or an options file. The standard reports include a pass/fail summary of code analysis and test results, a list of analyzed files, and a code coverage summary. The reports can be customized to include a listing of active static analysis checks, expanded test output with pass/fail status of individual tests, parameters of trend graphs for key metrics, and full code listings with color-coding of all code coverage results. Generated reports can be automatically sent via email, based on a variety of role-based filters. In addition to providing data directly to the developers responsible for the code flagged for defects, C++test sends summary reports to managers and team leads.

Efficient Team Deployment

C++test establishes an efficient process that ensures software verification tasks are ingrained into the team's existing workflow and automated—enabling the team to focus on tasks that truly require human intelligence. Defect review and correction are facilitated through automated task assignment and distribution. Each defect detected is prioritized, assigned to the developer who wrote the related code, and distributed to his or her IDE with full data and cross-links to code. To help managers assess and document trends, centralized reporting ensures real-time visibility into quality status and processes. This data also helps determine if additional actions are needed to satisfy internal goals or demonstrate regulatory compliance.

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 for DO-178B (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
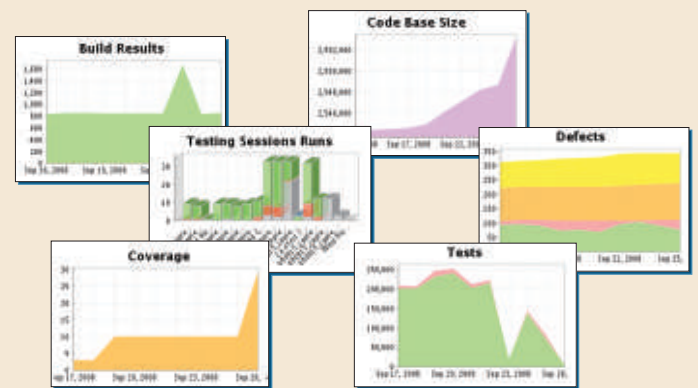

Dashboards track key development metrics

Embedded and Cross-Platform Development

As the software components in embedded systems are becoming increasingly critical, the attention to quality in embedded software increases across the board. Long-standing quality strategies such as testing with a debugger are no longer efficient or sufficient. To further complicate matters, many developers cannot readily run a test program in the actual deployment environment because they lack access to the final system hardware. To address these challenges, code quality needs to be realized throughout the development lifecycle—using a synergy of time-proven techniques for early defect prevention, assisted by automation for implementation and monitoring.
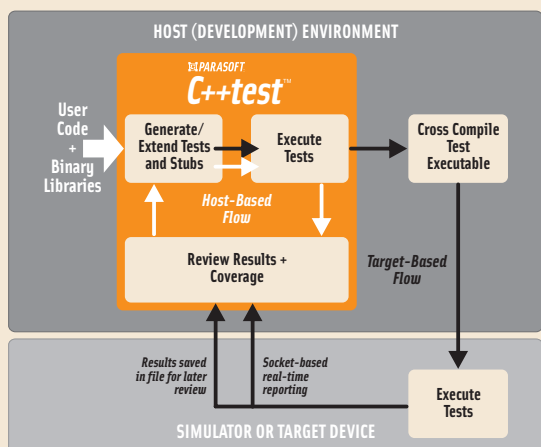
For highly quality-sensitive industries, such as avionics, medical, automobile, transportation, and industrial automation, the addition of Parasoft's Web-based audit and reporting system, with interactive Web-based dashboards and drill-down capability powered by a SQL database, enables an efficient and auditable quality process with complete visibility into compliance efforts.

HOST (DEVELOPMENT) ENVIRONMENT

PARASOFT
C++test™

User Code + Binary Libraries

Generate/ Extend Tests and Stubs → Execute Tests → Cross Compile Test Executable

Host-Based Flow

Review Results + Coverage

Target-Based Flow

Results saved in file for later review

Socket-based real-time reporting

Execute Tests

SIMULATOR OR TARGET DEVICE

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

Test on the Host, Simulator, and Target

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, as well as 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 liked 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.

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.

Supported Host Environments

Host Platforms
- Windows NT/2000/XP/2003/Vista/7
- Linux kernel 2.4 or higher with glibc 2.3 or higher and an x86-compatible processor
- Linux kernel 2.6 or higher with glibc 2.3 or higher and an x86_64-compatible processor
- Solaris 7, 8, 9, 10 and an UltraSPARC processor
- IBM AIX 5.3 and a PowerPC processor

IDEs
- Eclipse IDE for C/C++ Developers 3.2, 3.3, 3.4, 3.5, 3.6, 3.7
- Microsoft Visual Studio 2003, 2005, 2008, 2010 with Visual C++
- Wind River Workbench 2.6 or 3.0-3.3
- ARM Workbench IDE for RVDS 3.0, 3.1, 4.0, 4.1
- QNX Momentics IDE 4.5 (QNX Software Development Platform 6.4)
- Texas Instruments Code Composer Studio v4, v5

IDEs with Project Import Support
- ARM ADS 1.2
- Green Hills MULTI 4.0.x
- IAR Embedded Workbench 5.3/5.4
- Keil RealView MDK 3.40/uVision3
- Microsoft eMbedded Visual C++ 4.0
- Microsoft Visual Studio 6
- Texas Instruments Code Composer 3.1 and 3.3
- Wind River Tornado 2.0, 2.2

Host Compilers
- Windows
  - Microsoft Visual C++ 6.0, .NET (7.0), .NET 2003 (7.1), 2005 (8.0), 2008 (9.0), 2010 (10.0)
  - GNU and MingW gcc/g++ 2.95.x, 3.2.x, 3.3.x, 3.4.x
  - GNU gcc/g++ 4.0.x, 4.1.x, 4.2.x, 4.3.x, 4.4.x, 4.5.x
  - Green Hills MULTI for Windows x86 Native v4.0.x
- Linux (x86 target platform)
  - GNU gcc/g++ 2.95.x, 3.2.x, 3.3.x, 3.4.x, 4.0.x, 4.1.x, 4.2.x, 4.3.x, 4.4.x
- Linux (x86_64 target platform)
  - GNU gcc/g++ 3.3.x, 3.4.x, 4.0.x, 4.1.x, 4.2.x, 4.3.x, 4.4.x
- Solaris
  - Sun C++ 5.3 (Sun Forte C++ 6 Update 2), Sun C++ 5.5 (Sun ONE Studio 8), Sun C++ 5.6 (Sun ONE Studio 9), Sun C++ 5.7 (Sun ONE Studio 10), Sun C++ 5.8 (Sun ONE Studio 11), Sun C++ 5.9 (Sun ONE Studio 12)
  - GNU gcc/g++ 2.95.x, 3.2.x, 3.3.x, 3.4.x, 4.0.x, 4.1.x, 4.2.x, 4.3.x, 4.4.x, 4.5.x
  - Green Hills MULTI for SPARC Solaris Native v4.0.x
- AIX
  - IBM XL C/C++ compiler 8.0
  - GNU gcc/g++ 4.1.x

Target/Cross Compilers
- Altera (Linux hosted)
  - Nios GCC 2.9 (static analysis only)
  - NIOS II 5.1 GCC 3.4 (static analysis only)
- ARM (Windows hosted)
  - ARM RVCT 2.2, 3.x, 4.x
  - ARM ADS 1.2
- Cosmic (Windows hosted)
  - Cosmic Software 68HC08 C Cross Compiler V4.6.x (static analysis only)
- eCosCentric (Linux hosted)
  - GCC 3.4.x (static analysis only)
- Fujitsu (Windows hosted)
  - FR Family SOFTUNE C/C++ Compiler V6
- GNU (Windows, Linux, Solaris hosted)
  - gcc 2.9 - 4.5
- Green Hills
  - Windows hosted
    - Green Hills MULTI v5.1.x optimizing compilers for Embedded V800
  - Windows, Solaris hosted
    - Green Hills MULTI v4.0.x optimizing compilers
- IAR (Windows hosted)
  - IAR ANSI C/C++ Compiler 5.3x, 5.4x, 5.5x for ARM
  - IAR ANSI C/C++ Compiler 6.1x for ARM (C only)
- Keil (Windows hosted)
  - ARM/Thumb C/C++ Compiler, RVCT3.1 for uVision
  - ARM C/C++ Compiler, RVCT4.0 for uVision
  - C51 Compiler V8.18 (static analysis only)
- Microsoft (Windows hosted)
  - Microsoft Visual C++ for Windows Mobile 8.0, 9.0
  - Microsoft Embedded Visual C++ 4.0
- QNX (Windows hosted)
  - GCC 2.9.x, 3.3.x, 4.2.x, 4.4.x
- Renesas (Windows hosted)
  - Renesas SH SERIES C/C++ Compiler V9.03
- STMicroelectronics (Windows hosted)
  - ST20 (static analysis only)
  - ST40 (static analysis only)
- TASKING
  - Windows and Solaris hosted
    - 80C196 C Compiler v6.0 (static analysis only)
  - Windows hosted
    - TriCore VX-toolset C/C++ Compiler 2.5 (C only)
    - TriCore VX-toolset C/C++ Compiler 2.5, 3.3, 3.4, 3.5
- QNX (Windows hosted)
  - GCC 2.9.x, 3.3.x, 4.2.x, 4.4.x
- Renesas (Windows hosted)
  - Renesas SH SERIES C/C++ Compiler V9.03
- STMicroelectronics (Windows hosted)

- ST20 (static analysis only)
- ST40 (static analysis only)
- TASKING
  - Windows and Solaris hosted
    - 80C196 C Compiler v6.0 (static analysis only)
  - Windows hosted
    - TriCore VX-toolset C/C++ Compiler 2.5 (C only)
    - TriCore VX-toolset C/C++ Compiler 2.5, 3.3, 3.4, 3.5
- Texas Instruments (Windows hosted)
  - Windows hosted – CCS 4.x:
    - TMS320C6x C/C++ Compiler v6.1.x
    - TMS320C2000 C/C++ Compiler v5.2.x
    - TMS320C55x C/C++ Compiler v4.3
    - TMS320C54x C/C++ Compiler v4.2 (static analysis only)
    - MSP430 C/C++ Compiler v3.2.x
  - Windows hosted – CCS 3.x:
    - TMS320C6x C/C++ Compiler v5.1
    - TMS320C6x C/C++ Compiler v6.0
    - TMS320C2000 C/C++ Compiler v4.1 (static analysis only)
  - Solaris hosted:
    - TMS320C2x/C2xx/C5x Version 7.00 (static analysis only)
    - TMS320C6x C/C++ Compiler v. 4.3 (static analysis only)
    - TMS320C6x C Compiler v. 4.00 (static analysis only)
    - TMS320C6x C/C++ Compiler v. 5.1 (static analysis only)
- Wind River
  - Windows, Solaris, Linux hosted
    - GCC 2.96, 3.4.x, 4.1.x, 4.3.x
    - DIAB 5.0, 5.5, 5.6, 5.7, 5.8, 5.9
  - Windows hosted
    - GCC 3.3.x for VxWorks 653 (static analysis only)
    - EGCS 2.90

Build Management
- GNU make
- Sun make
- Microsoft nmake
- JAM
- Other build scripts that can provide an option of overriding a compiler via an environment variable

Source Control
- AccuRev SCM
- Borland StarTeam
- CVS
- Git
- IBM/Rational ClearCase
- Microsoft Team Foundation Server
- Microsoft Visual SourceSafe
- Perforce SCM
- Serena Dimensions
- Subversion (SVN)
- Telelogic Synergy

www.parasoft.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