

Integrated with Leading Embedded Development Environments including:

- > Green Hills MULTI[®]
- > WindRiver Tornado[®]
- > LynuxWorks™
- > TI Code Composer Studio™
- > Diab SingleStep™
- > Cosmic
- > TASKING™
- > Synopsys[®] ARC™
- > CodeWarrior™
- > Analog Devices Visual DSP++[®]
- > ST Microelectronics[®]
- > HighTec TriCore[®]
- > Microchip[®]
- > Paradigm
- > Renesas™
- > ARM[®] RVDS™
- > IAR Systems[™]
- > KEIL™
- > NEC
- > QNX[®]
- > Borland[®]
- > Mercury Computer Systems™

Highlights:

- > Compatible with UNIX, Linux, and Windows Compilers
- > Eliminates Need to Build Test Drivers and Stubs Manually
- > Integrated Code-Coverage Capabilities, including MC/DC
- > Supports Host, Simulator, or Embedded Target Testing
- > Automates Regression Testing
- > User-Configurable Compiler Interface
- > Supports DO-178B, ISO 26262, IEC 61508, FDA, IEC 62304, and CENELEC Test Requirements

VectorCAST/C++™

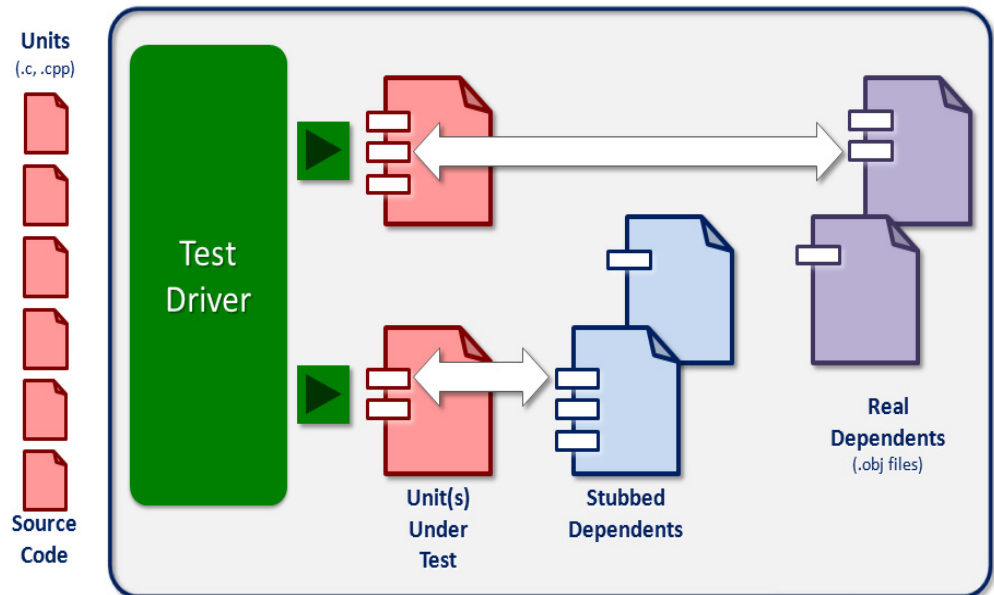
Unit/Integration Testing for C/C++

<What is VectorCAST/C++>

VectorCAST/C++ is an integrated software test solution that significantly reduces the time, effort, and cost associated with testing C/C++ software components necessary for validating safety and mission-critical embedded systems.

Automation includes:

- > Complete test-harness construction for unit and integration testing
- > Test execution from GUI or scripts
- > Integrations with best of breed requirements traceability and static analysis tools
- > Automatic test creation based on decision paths
- > User-defined tests for requirements-based testing
- > Regression testing
- > Test execution playback to assist in debugging
- > Code coverage analysis
- > Supports Agile and Test Driven Development Methods



VectorCAST/C++ builds test harnesses automatically

<Why VectorCAST/C++>

Generally, software-component testing requires generating at least one line of test code (in the form of stubs, drivers, and test data) for each line of application code to be tested. The necessity to create this “disposable” test software is the main reason why manual component testing is so expensive and inefficient. Test software not only has to be written but also has to be debugged to ensure that it performs as expected. With VectorCAST/C++, component testing can be performed without writing a single line of test code.

VectorCAST/C++ Capabilities

<How it Works>

VectorCAST/C++ parses your source code and invokes code generators to automatically create the test code required to construct a complete, executable test harness. Once the test harness is constructed, utilities can be used to build and execute test cases, show code covered, and report static measurements. Test-data is maintained independently of the test case, enabling automatic regression testing.

<Integrated Code Coverage>

Without a code-coverage tool, it is difficult to determine which portions of the source code have been exercised during testing. VectorCAST/C++ provides an integrated code-coverage utility that allows you to gauge the effectiveness of your component testing by reporting on the source-code statements or decision points exercised during individual or multiple test runs. Code-coverage data can also be shared with VectorCAST/Cover to produce combined coverage reports that reflect unit, integration, and system testing.

<Testing is Repeatable>

Once test cases have been developed, you can use VectorCAST/C++ to automatically run test cases against successive versions of the source code. The management of test execution and the cataloging of test results are automatic. Comparing results of the same test cases against new software versions, prior to system integration, results in fewer surprises caused by "one small change" to a software component.

<Supports Integration Testing>

Multiple units can be tested in a single VectorCAST/C++ test environment. This allows you to create complex test scenarios that stimulate multiple functions across multiple units.

<Compiler Integration>

All VectorCAST/C++ generated test-harness components are automatically compiled and linked, using your compiler. An interface to your compiler's debugger is also provided so that you can run test cases under control of the debugger.

<Test Driven Development>

VectorCAST/C++ supports Agile and Test Driven Development methodologies. Test case development now becomes the initial activity once the design is complete. This allows you to construct all unit tests prior to any application code being developed. Initially, unit tests will fail due to the absence of source code. But, with the incremental development of code for individual units, the unit tests will begin to pass. The unit test suite can then be regression tested automatically.

<Embedded Target Testing>

VectorCAST/C++ when used in conjunction with VectorCAST/RSP allows testing directly on your embedded target system. VectorCAST/RSP is integrated with the cross compiler and RTOS, making it the perfect tool for testing real-time applications. Tests may be developed in a host environment and then re-executed on an embedded target to verify target and cross-compiler performance.

Product Features:

- > Automatically creates complete test drivers and intelligent stubs for C/C++ code of any complexity: **no writing of test code**
- > Test drivers support complex test scenarios, including successive calls to multiple functions within the same test
- > Intelligent stubbing allows capture of inputs and control of outputs of any pre-defined or user-defined types
- > Tree-based graphical test-case editor makes creating and editing tests simple
- > Easily create test cases:
 - Test static, protected, and private functions
 - Construct class object instances of any complexity
 - Test Polymorphism and Dynamic Dispatching
 - Throw and catch exceptions of any type and value
 - Test complex class hierarchies
 - Test template instances individually
- > Unexpected exceptions and signals are caught and reported
- > Command Line Interface (CLI) allows scripting of full functionality

Easy-To-Use-GUI

- > **Test-Case Building** without writing test code. Values of parameters (of unit under test and stubs) and global data are defined by way of the GUI.
- > **Test Execution** does not require compilation for each test case.
- > **Pass/Fail** results of test cases are displayed in the GUI after test execution, along with color-coded pass/fail indicators.
- > **Code Coverage** is shown in a color-coded browser. Coverage levels are displayed for statement, branch, and MC/DC levels of coverage.
- > Execution can be performed on the host platform or target instruction-set simulator, or directly on an embedded target. Execution platform is controlled from the GUI.

