Built-in Test Configurations

This topic describes the preconfigured "built-in" Test Configurations that are included with C++test.

C++test includes a set of preconfigured "built-in" Test Configurations representing most common test scenarios. You can further customize these configurations as needed by copying and modifying the built-in configurations, or by creating new user-defined configurations from scratch. User-defined Test Configurations can be placed in the User-defined or Team category. User-defined Test Configurations are stored on the local machine and are available for all tests performed by the local C++test installation. Team Test Configurations are stored on the team's Team Server and can be accessed by all team members.

Static Analysis Group

This group includes universal static analysis test configurations. See Compliance Packs for test configurations that enforce coding standards

Test Configuration	Description
Recommended Rules	The default configuration of recommended rules. Covers most Severity 1 and Severity 2 rules. Includes rules in the Flow Analysis Fast configuration.
Flow Analysis Standard	Detects complex runtime errors without requiring test cases or application execution. Defects detected include using uninitialized or invalid memory, null pointer dereferencing, array and buffer overflows, division by zero, memory and resource leaks, and dead code. This requires a special Flow Analysis license option. See Introducing Built-in Flow Analysis Test Configurations for more details on Flow Analysis Test Configurations.
Flow Analysis Fast	The fast configuration uses "Shallowest" depth of analysis and runs faster than the standard and aggressive configurations. The fast configuration finds a moderate amount of problems and prevents violation number explosion. See Introducing Built-in Flow Analysis Test Configurations for more details on Flow Analysis Test Configurations.
Flow Analysis Aggressive	The aggressive option reports any suspicious code as a violation. See Introducing Built-in Flow Analysis Test Configurations for more details on Flow Analysis Test Configurations.
Effective C++	Checks rules from Scott Meyers' "Effective C++" book. These rules check the efficiency of C++ programs.
Effective STL	Checks rules from Scott Meyers' "Effective STL" book.
Modern C++ (11, 14 and 17)	Checks rules that enforce best practices for modern C++ standards (C++11, C++14, C++17).
Find Duplicated Code	Detects duplicated functions, code fragments, string literals, and #include directives.
Find Unused Code	Includes rules for identifying unused/dead code.
Metrics	Reports metrics statistics and detects metric values out of acceptable ranges.
Global Analysis	Checks the Global Static Analysis rules.
Sutter-Alexandrescu	Checks rules based on the book "C++ Coding Standards," by Herb Sutter and Andrei Alexandrescu.
The Power of Ten	Checks rules based on Gerard J. Holzmann's article "The Power of Ten - Rules for Developing Safety Critical Code." (http://spinroot.com/gerard/pdf/Power_of_Ten.pdf)

Compliance Packs

Compliance Packs include test configurations tailored for particular compliance domains to help you enforce industry-specific compliance standards and practices. See Compliance Packs Rule Mapping for information how the standards are mapped to C/C++test's rules.

Displaying compliance results on DTP

Some test configurations in this category have a corresponding "Compliance" extension on DTP, which allows you to view your security compliance status, generate compliance reports, and monitor the progress towards your security compliance goals. These test configurations require dedicated license features to be activated. Contact Parasoft Support for more details on Compliance Packs licensing.

See the "Extensions for DTP" section in the DTP documentation for the list of available extensions, requirements, and usage.

Aerospace Pack

Test Configuration	Description
Joint Strike Fighter	Checks rules that enforce the Joint Strike Fighter (JSF) program coding standards.
DO178C Software Level A Unit Testing	Executes unit tests with appropriate configuration of coverage metrics and reporting settings for DO178C Software Level A
DO178C Software Level B Unit Testing	Executes unit tests with appropriate configuration of coverage metrics and reporting settings for DO178C Software Level B
DO178C Software Level C and D Unit Testing	Executes unit tests with appropriate configuration of coverage metrics and reporting settings for DO178C Software Level C and D

Automotive Pack

Test Configuration	Description
AUTOSAR C++14 Coding Guidelines	Checks rules that enforce the AUTOSAR C++ Coding Guidelines (Adaptive Platform, version 17-10). This test configuration is part of Parasoft Compliance Pack solution that allows you to monitor compliance with industry standards using the "Compliance" extensions on DTP. It requires dedicated license features to be activated. Contact your Parasoft representative for details.
High Integrity C++	Checks rules that enforce the High Integrity C++ Coding Standard.
HIS Source Code Metrics	Checks metrics required by the Herstellerinitiative Software (HIS) group.
MISRA C 1998	Checks rules that enforce the MISRA C coding standards.
MISRA C 2004	Checks rules that enforce the MISRA C 2004 coding standards.
MISRA C++ 2008	Checks rules that enforce the MISRA C++ 2008 coding standards.
MISRA C 2012	Checks rules that enforce the MISRA C 2012 coding standards. This test configuration is part of Parasoft Compliance Pack solution that allows you to monitor compliance with industry standards using the "Compliance" extensions on DTP. It requires dedicated license features to be activated. Contact your Parasoft representative for details.
ISO26262 ASIL A Unit Testing	Executes unit tests with appropriate configuration of coverage metrics and reporting settings for ISO26262 ASIL A
ISO26262 ASIL B and C Unit Testing	Executes unit tests with appropriate configuration of coverage metrics and reporting settings for ISO26262 ASIL B and C
ISO26262 ASIL D Unit Testing	Executes unit tests with appropriate configuration of coverage metrics and reporting settings for ISO26262 ASIL D

Medical Devices Pack

Test Configuration	Description
Recommended Rules for FDA (C)	Checks rules recommended for complying with the FDA General Principles for Software Validation (test configuration for the C language).
Recommended Rules for FDA (C++)	Checks rules recommended for complying with the FDA General Principles for Software Validation (test configuration for the C++ language).

Security Pack

Test Configuration	Description

CWE-SANS Top 25 Most Dangerous Programming Errors	Checks for the 2011 CWE/SANS Top 25 Most Dangerous Software Errors— a list of the most widespread and critical errors that can lead to serious vulnerabilities in software. They are often easy to find, and easy to exploit. They are dangerous because they will frequently allow attackers to completely take over the software, steal data, or prevent the software from working at all. (http://cwe.mitre.org/top25/index.html) For more details, see 2011 CWE/SANS Top 25 Most Dangerous Software Errors Mapping.
OWASP Top 10 2017	Includes rules that find issues identified in OWASP's Top 10 standard.
Payment Card Industry Data Security Standard	Checks rules for the security issues referenced in section 6 of the Payment Card Industry Data Security Standard (PCI DSS) (https://www.pcisecuritystandards.org/security_standards/pci_dss.shtml) Issues detected include input validation (to prevent cross-site scripting, injection flaws, malicious file execution, etc.) and validation of proper error handling.
Security Rules	Checks rules designed to prevent or identify security vulnerabilities.
SEI CERT C Coding Guidelines	Checks rules and recommendations for the SEI CERT C Coding Standard. This standard provides guidelines for secure coding. The goal is to facilitate the development of safe, reliable, and secure systems by, for example, eliminating undefined behaviors that can lead to undefined program behaviors and exploitable vulnerabilities.
SEI CERT C Rules	Checks rules for the SEI CERT C Coding Standard. This standard provides guidelines for secure coding. The goal is to facilitate the development of safe, reliable, and secure systems by, for example, eliminating undefined behaviors that can lead to undefined program behaviors and exploitable vulnerabilities. This test configuration is part of Parasoft Compliance Pack solution that allows you to monitor compliance with industry standards using the "Compliance" extensions on DTP. It requires dedicated license features to be activated. Contact your Parasoft representative for details.
SEI CERT C++ Rules	Checks rules for the SEI CERT C++ Coding Standard. This standard provides guidelines for secure coding. The goal is to facilitate the development of safe, reliable, and secure systems by, for example, eliminating undefined behaviors that can lead to undefined program behaviors and exploitable vulnerabilities. This test configuration is part of Parasoft Compliance Pack solution that allows you to monitor compliance with industry standards using the "Compliance" extensions on DTP. It requires dedicated license features to be activated. Contact your Parasoft representative for details.
UL 2900	Includes rules that find issues identified in the UL-2900 standard.

Unit Testing Group

Test Configuration	Description
File Scope> Build Test Executable (File Scope)	Builds test executable for "trial builds." *Only the selected file(s) will be instrumented.*
File Scope> Collect Stub Information (File Scope)	Collects symbols data to populate the Stubs view. *Only the selected file(s) will be instrumented.*
File Scope> Debug Unit Tests (File Scope)	Executes unit tests under the debugger. *Only the selected file(s) will be instrumented.*
File Scope> Generate Stubs (File Scope)	Generates stubs for missing function and variable definitions. *Only the selected file(s) will be instrumented.*
File Scope> Run Unit Tests	Executes the available test cases. *Only the selected file(s) will be instrumented.*
Build Test Executable	Builds test executable for "trial builds." *All project files will be instrumented.*
Collect Stub Information	Collects symbols data to populate the Stubs view. *All project files will be instrumented.*
Debug Unit Tests	Executes unit tests under the debugger. *All project files will be instrumented.*

Generate Regression Base	Generates a baseline test suite that captures the project code's current functionality; to detect changes from this baseline, you run your evolving code base against this test suite on a regular basis. Outcomes are automatically verified.
Generate Stubs	Generates stubs for missing function and variable definitions. *All project files will be instrumented.*
Generate Test Suites	Generates test suites (without generating test cases) for the selected resources.
Generate Unit Tests	Generates unit tests for the selected resources.
Run Unit Tests	Executes the available test cases. *All project files will be instrumented.*
Run Unit Tests with Memory Monitoring	Executes the available test cases and collects information about memory problems. *All project files will be instrumented.*

Application Monitoring Group

Test Configuration	Description
Build Application with Coverage Monitoring	Builds the tested application with coverage monitoring enabled.
Build Application with Full Monitoring	Builds the tested application with coverage and memory monitoring enabled.
Build Application with Memory Monitoring	Builds the tested application with memory monitoring enabled.
Build and Run Application with Coverage Monitoring	Builds and executes the tested application with coverage monitoring enabled.
Build and Run Application with Full Monitoring	Builds and executes the tested application with coverage and memory monitoring enabled.
Build and Run Application with Memory Monitoring	Builds and executes the tested application with memory monitoring enabled.

Embedded Systems Group

Test Configuration	Description
Window Mobile> Build Test Executable for Windows Mobile	Builds a test executable that you need to manually transfer to the target device and run. This Test Configuration is very similar to the "Build Test Executable" Test Configuration; the only difference is that it is configured to use an external storage card to generate post-run artifacts (coverage and results). See Windows Mobile Support for details.
Window Mobile> Build and Run Test Executable for Pocket PC	Builds the test executable, then deploys it to the emulator and runs it. After execution completes, you need to close the emulator to prompt C++test to read and display test results. See Windows Mobile Support for details.
Window Mobile> Build and Run Test Executable for Smartphone	Builds the test executable, then deploys it to the emulator and runs it. After execution completes, you need to close the emulator to prompt C++test to read and display test results. See Windows Mobile Supportfor details.
Window Mobile> Build and Run Test Executable for Windows Mobile or Windows CE Using ActiveSync	Builds the test executable, then deploys it to the emulator and runs it. ActiveSync is used as a communication channel. To use this flow, both host and target machines must support ActiveSync. The target can be a real device connected in ways supported by ActiveSync, or it can be an emulator. See Windows Mobile Support for details.

Utilities Group

Test Configuration	Description
Load Test Results (File)	Used to collect test results via the file channel. By default, this configuration assumes that logs are located inside `${cpptest:testware_loc}`. If needed, you can customize this location to any file system location that can be accessed from the C++test GUI.
Load Test Results (Sockets)	Used for "on the fly" collection of test results sent through TCP/IP sockets. It starts a java utility program to listen to and capture test results. You can customize the port numbers for test and coverage results. Port numbers are defined with the `results_port` and `coverage_port` properties.

Extract Library Symbols	Used to extract a list of symbols from external libraries (or object files). It should be used whenever C++test's standard algorithm for collecting information about symbols from binaries is not sufficient. For example if you use a Wind River DKM type of project, you may want to have all symbols from the VxWorks image collected in this way. You will probably need to enter the location of the binaries you want to extract symbols from, as well as the name of the nm-like utility that can be used to dump the content of library /object file.
Generate Stubs Using External Library Symbols	Used to generate stubs after the "Extract Library Symbols" Test Configuration has been run. It assumes that a file with a list of symbols from external libraries is stored in the project temporary data.
Load Application Coverage	Used to import the coverage data collected with the `cpptestcc` coverage tool into your IDE; see Collecting Application Coverage with cpptestcc.
Load Archived Results	Used to load the archived results into C/C++test; see Merging Results from Multiple Test Runs.

Code Review Group

Name	Scope	Code Review
Pre-Commit	Only files added or modified locally	For teams who want to review code *before* it is committed to source control. To use this Test Configuration, the Code Review Preference **Show user assistant during scanner run** setting must be enabled so that the author can designate the appropriate reviewer(s). See the Code Review for details.
Post-Commit	All project files modified in the previous day	For teams who want to review code after it is committed to source control. This Test Configuration must be duplicated and customized prior to use (e.g, to specify author-reviewer mappings). See Code Review for details.

See Configuring Test Configurations and Rules for Policies to learn how to develop custom Test Configurations that are tailored to your projects and team priorities.

Compliance Packs Rule Mapping

This section includes rule mapping for the OWASP and CWE standars. The mapping information for other standards is available in the PDF rule mapping files shipped with Compliance Packs.

OWASP Top 10 - 2017 Mapping

OWASP Category	CWE ID	Parasoft Rule IDs
A1 Injection	CWE-77: Command Injection	• BD-SECURITY-TDCMD
A1 Injection	CWE-89: SQL Injection	• BD-SECURITY-TDSQL
A3 Sensitive Data Exposure	CWE-326: Weak Encryption	• SECURITY-37
A3 Sensitive Data Exposure	CWE-327: Use of a Broken or Risky Cryptographic Algorithm	• SECURITY-02 • SECURITY-28 • SECURITY-37
A5 Broken Access Control	CWE-22: Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	• BD-SECURITY-TDFNAMES
A6 Security Misconfiguration	CWE-391: Unchecked Error Condition	• BD-PB-ERRNO
A6 Security Misconfiguration	CWE-396: Declaration of Catch for Generic Exception	• EXCEPT-17

A10 Insufficient Logging & Monitoring	CWE-223: Omission of Security-relevant Information	• SECURITY-14 • SECURITY-15

2011 CWE/SANS Top 25 Most Dangerous Software Errors Mapping

CWE ID	CWE Name	Parasoft ID	Parasoft Name
CWE-89	Improper Neutralization of Special Elements used in an SQL Command ('SQL Injection')	BD-SECURITY-TDSQL	Protect against SQL injection
CWE-78	Improper Neutralization of Special Elements used in an OS Command ('OS Command Injection')	BD-SECURITY-TDCMD	Protect against command injection
CWE-120	Buffer Copy without Checking Size of Input ('Classic Buffer Overflow')	BD-PB-OVERFFM	Avoid buffer overflow due to defining incorrect format limits
		BD-PB-OVERFNZT	Avoid overflow due to reading a not zero terminated string
		BD-PB-OVERFWR	Avoid overflow when writing to a buffer
		BD-SECURITY-OVERFWR	Avoid buffer write overflow from tainted data
CWE-22	Improper Limitation of a Pathname to a Restricted Directory ('Path Traversal')	BD-SECURITY-TDFNAMES	Protect against file name injection
CWE-676	Use of Potentially Dangerous Function	PB-37	The unbounded functions of library shall not be used
		SECURITY-11	Avoid using unsecured shell functions that may be affected by shell metacharacters
		SECURITY-12	Avoid using unsafe string functions which may cause buffer overflows
		SECURITY-13	Avoid using unsafe string functions that do not check bounds
		SECURITY-14	Do not use scanf and fscanf functions without specifying variable size in format string
		SECURITY-16	Never use gets()
		SECURITY-22	Do not use mbstowcs() function
		SECURITY-30	Avoid using 'getpw' function in program code
		SECURITY-31	Do not use 'cuserid' function
CWE-327	Use of a Broken or Risky Cryptographic Algorithm	SECURITY-02	Avoid functions which use random numbers from standard C library
		SECURITY-28	Standard random number generators should not be used to generate randomness for security reasons
		SECURITY-37	Do not use weak encryption functions
CWE-131	Incorrect Calculation of Buffer Size	BD-PB-ARRAY	Avoid accessing arrays out of bounds
		BD-PB-OVERFRD	Avoid overflow when reading from a buffer
		BD-SECURITY-ARRAY	Avoid tainted data in array indexes
		MRM-45	Do not use sizeof operator on pointer type to specify the size of the memory to be allocated via 'malloc', 'calloc' or 'realloc' function
CWE-134	Uncontrolled Format String	SECURITY-05	Avoid using functions printf/wprintf with only one variable parameter
		SECURITY-08	Avoid using functions fprintf/fwprintf with only two parameters, when second parameter is a variable
CWE-190	Integer Overflow or Wraparound	BD-SECURITY-INTOVERF	Protect against integer overflow/underflow from tainted data
		MISRA-051	Evaluation of constant unsigned integer expressions should not lead to wrap-around