Coverity 8.0 Command and Ant Task Reference

Reference for Coverity Analysis, Coverity Platform, and Coverity Desktop.

Table of Contents

Coverity Analysis Commands

Name

cov-analyze, cov-analyze-java Analyze an intermediate directory for quality and security defects.

Synopsis

`cov-analyze` --dir <intermediate_directory> [OPTIONS]

`cov-analyze-java` --dir <intermediate_directory> [OPTIONS]

Description

The `cov-analyze` command runs checkers on captured code in an intermediate directory and stores analysis results in that directory, which is specified with `--dir`. This command typically follows cov-build and precedes cov-commit-defects invocations on the same intermediate directory. The `cov-analyze-java` command is a deprecated form of `cov-analyze` that only analyzes Java code, like `cov-analyze --java`. Though `cov-analyze` does not report defects in Java and .NET bytecode, nor in some forms of source code not written by a person, the command does run an analysis of them for the benefit of finding interprocedural defects in editable source code.

A log file (`analysis-log.txt`) with information about the checkers used in the analysis, including notices of crashes, is located in the following directory: `<intermediate_directory>/output`

Note

If you get a fatal `No license found` error when you attempt to run this command, you need to make sure that `license.dat` was copied correctly to `<install_dir>/bin`.

On some Windows platforms, you might need to use administrative privileges when you copy the Coverity Analysis license to `<install_dir>/bin`. Due to file virtualization in some versions of Windows, it might look like `license.dat` is in `<install_dir>/bin` when it is not.

Typically, you can set the administrative permission through an option in the right-click menu of the executable for the command interpreter (for example, `Cmd.exe` or Cygwin) or Windows Explorer.

Options

--aggressiveness-level <level>

Enables a set of checker flags and `cov-analyze` options that cause Coverity Analysis to make more aggressive assumptions during analysis. Higher levels report more defects, and the analysis time increases. Values for `level` are `low`, `medium`, or `high`. Default is `low`.

Starting in version 7.0, this option applies to all programming languages that undergo analysis with `cov-analyze`. If a checker option applies to multiple languages, the aggressiveness level tuning will apply to that option for all supported languages. Changes to checker options that do not apply to a given language have no effect or related warnings.

The aggregate false positive rate for all checkers that are not parse warnings checkers is approximately 50% higher for `medium`, and 70% higher for `high`. Different aggressiveness levels do not change the rate of false positives that parse warning checkers report. A higher aggressiveness level for parse warning checkers enables more warnings for less severe defects.

The value `low` uses the default for all checkers and options. For a list of checker option defaults, see Checker Option Defaults .

The value `medium` uses the settings at the `low` level with the following overrides:

```
            --enable-parse-warnings:true [C/C++]
            --no-field-offset-escape:true [C/C++]
            BAD_ALLOC_STRLEN:report_plus_any:true [C/C++]
            CALL_SUPER:threshold:.55 [C#, Java]
            CHECKED_RETURN:error_on_use:true [C/C++, Java]
            CHECKED_RETURN:stat_threshold:55 [C/C++, Java]
            CONSTANT_EXPRESSION_RESULT:report_bit_and_with_zero:true [C/C++, C#,
 Java]
            CONSTANT_EXPRESSION_RESULT:report_constant_logical_operands:true [C/C+
+, C#, Java]
            FORWARD_NULL:aggressive_derefs:true [C/C++, C#, Java]
            FORWARD_NULL:deref_zero_errors:true [C/C++, C#, Java]
            FORWARD_NULL:track_macro_nulls:true [C/C++]
            INFINITE_LOOP:allow_asm:true [C/C++]
            INFINITE_LOOP:allow_pointer_derefs:true [C/C++, C#, Java]
            INFINITE_LOOP:report_no_escape:true [C/C++, C#, Java]
            NO_EFFECT:self_assign_to_local:true [C/C++]
            NO_EFFECT:unsigned_enums:true [C/C++]
            NULL_RETURNS:allow_unimpl:true [C/C++, C#, Java]
            NULL_RETURNS:stat_bias:10 [C/C++, C#, Java]
            NULL_RETURNS:stat_threshold:50 [C/C++, C#, Java]
            NULL_RETURNS:suppress_under_related_conditional:false [C/C++, C#, Java]
            OVERFLOW_BEFORE_WIDEN:check_macros:true [C/C++]
            OVERFLOW_BEFORE_WIDEN:check_nonlocals:true [C/C++, C#, Java]
            OVERFLOW_BEFORE_WIDEN:report_intervening_widen:true
            OVERRUN:report_underrun:true [C/C++]
            PW.DECLARED_BUT_NOT_REFERENCED [C/C++]
            REGEX_CONFUSION [Java]
            RESOURCE_LEAK:allow_cast_to_int:true [C/C++]
            RESOURCE_LEAK:allow_main:true [C/C++]
            RESOURCE_LEAK:allow_overwrite_model:true [C/C++]
            RESOURCE_LEAK:allow_unimpl:true [C/C++]
            RESOURCE_LEAK:track_fields:true [C/C++]
            SYMBIAN.CLEANUP_STACK:bad_pop:true [C++]
            SYMBIAN.CLEANUP_STACK:infer_allocs:true [C++]
            SYMBIAN.CLEANUP_STACK:multiple_pushes:true [C++]
            SYMBIAN.NAMING:report_LC_errors:true [C++]
            UNINIT:check_arguments:true [C/C++]
            UNINIT:check_mayreads:true [C/C++]
            UNINIT:enable_deep_read_models:true [C/C++]
            UNINIT:enable_parm_context_reads:true [C/C++]
            UNINIT:enable_write_context:true [C/C++]
            UNUSED_VALUE:report_dominating_assignment [C/C++, C#, and Java]
            UNUSED_VALUE:report_unused_final_assignment [C/C++, C#, and Java]
            UNUSED_VALUE:report_unused_initializer [C/C++, C#, and Java]
            USELESS_CALL:include_current_object_call_sites:true [C/C++, C#, Java]
            USELESS_CALL:include_macro_call_sites_fn:true [C/C++]
```

```
            USELESS_CALL:include_macro_call_sites_plain:true [C/C++]
```

The value `high` uses all the `medium` level settings, as well as the following:

```
          CONSTANT_EXPRESSION_RESULT:report_bit_and_with_zero_in_macros:true [C/C+
+]

 CONSTANT_EXPRESSION_RESULT:report_constant_logical_operands_in_macros:true [C/C+
+]
          CONSTANT_EXPRESSION_RESULT:report_unnecessary_op_assign:true [C/C++, C#,
 Java]
          FORWARD_NULL:very_aggressive_derefs:true [C/C++, C#, Java]
          INFINITE_LOOP:suppress_in_macro:false [C/C++]
          INTEGER_OVERFLOW:enable_all_overflow_ops:true [C/C++]
          INTEGER_OVERFLOW:enable_deref_sink:true [C/C++]
          NESTING_INDENT_MISMATCH:report_bad_indentation:true [C/C++, C#, Java]
          NO_EFFECT:self_assign_in_macro:true [C/C++]
          NULL_RETURNS:stat_threshold:0 [C/C++, C#, Java]
          OVERFLOW_BEFORE_WIDEN:check_bitwise_operands:true [C/C++, C#, Java]
          OVERFLOW_BEFORE_WIDEN:check_types:.* [C/C++]
          OVERFLOW_BEFORE_WIDEN:ignore_types:^$ [C/C++]
          OVERRUN:check_nonsymbolic_dynamic [C/C++]
          PW.ALREADY_DEFINED [C/C++]
          PW.BAD_INITIALIZER_TYPE [C/C++]
          PW.BAD_RETURN_VALUE_TYPE [C/C++]
          PW.CLASS_WITH_OP_DELETE_BUT_NO_OP_NEW [C/C++]
          PW.CLASS_WITH_OP_NEW_BUT_NO_OP_DELETE [C/C++]
          PW.ILP64_WILL_NARROW [C/C++]
          PW.INCOMPATIBLE_ASSIGNMENT_OPERANDS [C/C++]
          PW.INCOMPATIBLE_OPERANDS [C/C++]
          PW.INCOMPATIBLE_PARAM [C/C++]
          PW.INTEGER_TRUNCATED [C/C++]
          PW.MIXED_ENUM_TYPE [C/C++]
          PW.NESTED_COMMENT [C/C++]
          PW.NO_CORRESPONDING_DELETE [C/C++]
          PW.NO_CORRESPONDING_MEMBER_DELETE [C/C++]
          PW.NO_CTOR_BUT_CONST_OR_REF_MEMBER [C/C++]
          PW.NON_CONST_PRINTF_FORMAT_STRING [C/C++]
          PW.NONSTD_VOID_PARAM_LIST [C/C++]
          PW.NOT_COMPATIBLE_WITH_PREVIOUS_DECL [C/C++]
          PW.POINTER_CONVERSION_LOSES_BITS [C/C++]
          PW.SET_BUT_NOT_USED [C/C++]
          RESOURCE_LEAK:allow_address_taken:true [C/C++]
          RESOURCE_LEAK:allow_constructor:true [C/C++]
          RESOURCE_LEAK:allow_template:true [C/C++]
          RESOURCE_LEAK:allow_virtual:true [C/C++]
          SYMBIAN.CLEANUP_STACK:aliases_as_free:true [C++]
          TAINTED_STRING:paranoid_format:true [C/C++]
          UNCAUGHT_EXCEPT:report_all_fun:true [C++]
          UNINIT:allow_unimpl:true [C/C++]
          UNINIT:check_malloc_wrappers:true [C/C++]
          UNINIT_CTOR:allow_unimpl:true [C/C++]
```

```
          UNREACHABLE:report_unreachable_in_macro:true [C/C++]]
          WEAK_PASSWORD_HASH:report_weak_hashing_on_all_strings:true [Java Web App
Security]
```

--all
Enables almost all checkers that are disabled by default (exceptions are noted below). Using this option is equivalent to using all of the following options:

- `--concurrency`
- `--enable-parse-warnings`
- `--enable PARSE_ERROR`
- `--enable STACK_USE`
- `--preview`
- `--security`

COM checkers (Windows only)
On Windows systems, the `--all` option enables all preview COM checkers; however, on other systems you must explicitly use the `--enable` for the analysis to run them.

Exceptions
The following checkers are disabled by default, and the `--all` option *does not* turn them on:

- Rule checkers (which are enabled with the `--rule` option).
- Symbian checkers (which are enabled with the `--symbian` option).
- ENUM_AS_BOOLEAN
- HFA
- MISRA_CAST
- SECURE_CODING
- USER_POINTER
- Web application security checkers (such as XSS) are not affected by this option. To enable them, see `--webapp-security`, `--webapp-config-checkers`, and `--webapp-security-preview`.

Default checkers are enabled by default and are therefore unaffected by this option.

For information about enabling individual checkers, see the `--enable` option.

To find out whether a checker can be enabled with this option, see the --list-checkers option.

--allow-unmerged-emits
By default, the analysis fails if an intermediate directory contains emits of builds from multiple hosts. Specify this option to disable error checking and permit the analysis to continue in these cases.

If you use `cov-manage-emit add-other-hosts` to associate all emit repositories in the current intermediate directory with the current host, `--allow-unmerged-emits` is not needed to continue the analysis.

--append
Append defects from the last analysis run to the defects from this run.

By default, each analysis run includes individual checker-result files, the analysis summary file, and a metrics file. The `--append` option adds analysis results to the individual checker-result and summary files, but leaves the metrics file unchanged. So when you use the `--append` option, the metrics file will reflect the initial analysis without incremental analysis results for subsequent analyses that use the `--append` option.

The `--append` option is intended for appending issues found by custom Extend SDK checkers (see *Coverity Extend SDK 8.0 Checker Development Guide*) and for importing issues by the `cov-import-results` and `cov-import-msvsca` commands. This option does not allow multiple `cov-analyze` or `cov-analyze-java` commands with standard checkers to write results into the same intermediate directory. To analyze a mixed-language code base, use a single `cov-analyze` invocation.

When this option is used with the --output-tag option, `--append` applies to the output location that is specified through `--output-tag`.

--checker-option <checker_name>:<option>[:<option_value>], -co <checker_name>:<option>[:<option_value>]
Passes a checker option. Checker options and their default values are documented in the *Coverity 8.0 Checker Reference* .

Example:

```
INFINITE_LOOP:report_no_escape:true
```

Starting in version 7.0, when you specify the value of a checker option for a checker that supports the analysis of multiple languages, the value that you specify will apply to all languages to which that checker option applies. For example, if you set the `stat_threshold` to NULL_RETURNS, and you run an analysis on C/C++ and C# code bases, the value you set for that option will apply to both languages. If you do not set the value, the checker will use the default values for those options, which in a very limited number of cases can vary by language.

Some checker options are language-specific, such as `FORWARD_NULL:dynamic_cast`. This option is only available for (and can only apply to) C/C++ even though the FORWARD_NULL checker supports multiple languages.

--code-version-date <date>
For Test Advisor and Desktop Analysis, use this option to specify the date of the source code. This date is used for impact analysis. If possible, it is recommended to use the SCM checkout date of the code being analyzed.

If this option is not specified, the products will use the latest invocation timestamp of `cov-build` or `cov-capture`, as stored in the intermediate directory. If that is not available, then the current date and time are used instead.

This option is required when analyzing historical versions of your source code.

See Accepted date/time formats for proper formatting of the `<date>` argument.

Note

So that Test Advisor can correctly interpret function history used for impact analysis, please ensure that EITHER:

1. All machines used to do analysis have the same timezone setting.
2. A timezone is specified for all dates in the policy file and the `--code-version-date` argument. For example, use `2010-01-01 00:00-08:00` instead of `2010-01-01`.

--command <checker_pathname>
[Extend SDK analysis option] Uses a Extend SDK checker at the specified path name.

--concurrency
[C/C++ analysis option] Enables C/C++ production-level, concurrency checkers that are disabled by default. This option does not enable preview-level concurrency checkers.

For a list of concurrency checkers that you can enable with this option, see list-checkers.

--cpp
[C/C++ analysis option] Restricts the analysis to C/C++ code (even if code from other languages has been captured through a build to the `emit` subdirectory of the same intermediate directory). It is an error to combine the use of this option with the `--java` or `--cs` options.

--cs
[C# analysis option] Restricts the analysis to C# code (even if code from other languages has been captured through a build to the `emit` subdirectory of the same intermediate directory). It is an error to combine the use of this option with the `--java` or `--cpp` options.

--cxx
This option is deprecated and no longer has any effect. The corresponding checkers BAD_OVERRIDE, CTOR_DTOR_LEAK, DELETE_ARRAY, INVALIDATE_ITERATOR, PASS_BY_VALUE, UNCAUGHT_EXCEPT, UNINIT_CTOR, WRAPPER_ESCAPE, and, on Windows, COM.BAD_FREE and COM.BSTR.CONV, are now enabled by default. Checkers that can only find defects in C++ code automatically do not run on C code. Note that PASS_BY_VALUE can find defects in C code. If you were using `--disable-default --cxx`, replace it with individual `--enable` options.

--da-max-mem
[Java Web Application Security option] Sets the JVM heap size of the VM that is running the dynamic analyzer, a component of the Java Web Application Security analysis. The option accepts an integer that specifies a number of megabytes (MB). The default value is 1024.

--dc-config <file.json>
Identifies a JSON file for one or more custom DC.CUSTOM_* (custom Don't Call) checkers that you intend to run in the analysis. For details, see DC.*CUSTOM_CHECKER* in the *Coverity 8.0 Checker Reference*.

Note that use of this option enables all the DC.CUSTOM_* checkers that are configured in the JSON file. You can disable them individually with --disable <checker-name>. The --disable-default option will disable all of them.

--derived-model-file <derived_file.xmldb>
[Deprecated as of version 7.7.0] This option will be removed and replaced in a future release. Use --model-file instead.

--dir <intermediate_directory>
Pathname to an intermediate directory that is used to store the results of the build and analysis.

--disable <checker_name>, -n <checker>
Disables a checker. This option can be specified multiple times. See also `--list-checkers` and `--disable-default`. .

To find out whether a checker is enabled or disabled by default, see the `--list-checkers` option.

You can also use this option to disable individual FindBugs bug patterns. To specify a pattern, you need to add an FB prefix to it. For example:

```
--disable FB.DM_EXIT
```

To disable FindBugs bug patterns, you can also use `--disable-fb` or `--fb-exclude`.

--disable-default
Disables default checkers. This option is useful if you want to disable all default checkers and then enable only a few with the `--enable` option.

For a list of checkers that are disabled through this option, see the `--enable` option documentation for the `cov-analyze` command.

--disable-fnptr
[C/C++ analysis option] Disables analysis of calls to function pointers for defects. See also `--enable-fnptr`.

--disable-jshint
[JavaScript option] Disables JSHint analyses. The JSHint analysis is disabled by default.

Do not specify both --enable-jshint and `--disable-jshint` on the same command line.

--disable-parse-warnings
[C/C++ analysis option] Disables all parse warnings, and override other arguments that might have enabled them, such as `--all` or `--enable-parse-warnings`. The order of command-line options is irrelevant; the `--disable-parse-warnings` option takes precedence.

--enable <checker_name>, -en <checker>
Enables a checker that is not otherwise enabled by default. The checker name is case insensitive. This option will enable a checker for all languages supported by the checker. Note that default enablement of a given checker can vary by language.

Checkers are enabled by name, so related checkers such as MISSING_LOCK and GUARDED_BY_VIOLATION are enabled independently. The checker name is case insensitive. You can specify this option multiple times. See also `--list-checkers` and `--disable-default`.

Unlike the `--disable` option, this option does not work with FindBugs bug patterns.

If you enable a checker for which you do not have the appropriate license, and then attempt to run an analysis, you will receive an error. The Quality Advisor license covers all quality and FindBugs checkers. The Coverity Security Advisor license covers all Web Application Security checkers.

--enable-callgraph-metrics
Creates `<intermediate_directory>/output/callgraph-metrics.txt` and `<intermediate_directory>/output/checked-return.csv`.

The `callgraph-metrics.txt` file has information about which functions are analyzed. The file lists whether a function is implemented, which means it is analyzed, or whether a function is unimplemented, which means that it is not analyzed. A model is used if it is available. It also shows the number of callers for each function.

The `checked-return.csv` file stores information on the percentage of times that each the return value of each function is checked. This information can help you understand situations where the statistical checkers report different defects in local builds than they do in full builds.

For details, see *Coverity Analysis 8.0 User and Administrator Guide*.

Starting in version 7.0, applies to all programming languages.

--enable-constraint-fpp
Enables additional filtering of potential defects by using an additional false-path pruner (FPP). This option can increase the analysis time up to 20% (normally much less), but decrease the number of false positives that occur along infeasible paths. Because this FPP uses a different method for pruning false positives, it is possible that a very small number of true positives are pruned as well.

Note that use of this option requires an *additional* 200MB of memory per worker.

Starting in version 7.0, this option applies to C/C++, C#, and Java.

--enable-default
Enables all default checkers. This option takes precedence over the `--disable-default` option or the `--test-advisor` option. That is, if this option is specified then all default checkers are enabled regardless of the use of those options. However, individual checkers can still be disabled using the `--disable` option.

--enable-exceptions
Enables exceptional control flow analysis for C++. If specified, this option will report the following type of resource leak as a defect:

```
                    bool maybe;
          void test1() {
```

```
  int *x = new int;
  if (maybe) {
    throw 0; // x is leaked
  }
  delete x;
}
```

By default, the analysis ignores the exception type `std::bad_alloc` because some applications might not be designed to handle out-of-memory scenarios. If you specify `--enable-exceptions --handle-badalloc`, the analysis will report the following example as a defect. The example leaks memory if `new char` throws a `std::bad_alloc` exception:

```
void test() {
  int *x = new int;
  char *y = new char; // Leaks 'x' if this throws std::bad_alloc
  delete y;
  delete x;
}
```

Note that defects are not limited to leaks. For example, the FORWARD_NULL checker finds the following defect:

```
int *global;

void foo() {
    int *y = 0;
    try {
        // if std::bad_alloc is thrown, y remains null
        global = new int;
        y = global;
    } catch (...) {
        // empty
    }
    *y = 1; //FORWARD_NULL defect.
}
```

This option is disabled by default for C++ but enabled by default for Java and C#.

See also, --handle-badalloc.

--enable-fnptr
[C/C++ analysis option] Enables analysis of calls to function pointers for defects. By default, calls through function pointers are not used by the analysis engine for interprocedural analysis. When specified, this option allows up to 100 function resolutions for any function pointer. If that limit is exceeded, the analysis engine reverts to the default behavior.

When using this option, the analysis time typically increases by approximately 20%. However, the false positive rate might increase. See also `--disable-fnptr`.

This is a option.

--enable-jshint
[JavaScript option] Enables the JSHint analysis of captured JavaScript source code *except for* minified source files (see JSHINT.* in the *Coverity 8.0 Checker Reference* for details). Note that running a JSHint analysis requires an additional pass over all analyzed JavaScript code, which can significantly increase overall analysis time. The JSHint analysis is disabled by default.

JSHint reports a large number of defects, unless a custom configuration is used to trim down the results or the code is highly polished.

If you want to run the JSHint analysis with a custom `.jshintrc` configuration file, use --use-jshintrc. Otherwise, the analysis will use the Coverity default configuration file in `jshint/config.json` and ignore any `.jshintrc` files in your source tree.

Do not specify both --disable-jshint and `--enable-jshint` on the same command line.

--enable-parse-warnings
[C/C++ analysis option] Enables parse warnings, recovery warnings, and semantic warnings that are produced by the `cov-build` command so that they appear as defects in Coverity Connect. See also `--parse-warnings-config`.

This option is set automatically if the `--aggressiveness-level` option is set to `medium` (or to `high`).

--enable-single-virtual
Enables single, virtual-call resolution. By default, a C++ analysis treats all virtual functions as unimplemented, whereas full virtual call resolution is enabled by default for Java and C# analyses. When this option is enabled, interprocedural analysis across virtual calls takes place when the analysis engine finds only one implementation of a virtual function. When the analysis engine finds more than one implementation, it assumes that the virtual function is unimplemented. Do not specify this option if you specify the `--enable-virtual` option.

A C++ analysis can take longer than the default analysis because the analysis engine looks at implementations of virtual functions, which can result in more defect reports. Though this using this option might expedite Java and C# analyses, it also significantly affects results for interprocedural checkers.

Specify this option, or `--enable-virtual`, to enable interprocedural analysis of Apple Block invocations in C and C++ code.

Starting in version 7.0, applies to all programming languages.

--enable-test-metrics
This option creates test metrics data files (in the intermediate directory) that can be subsequently committed to Coverity Connect for storage. This option can be specified with other `cov-analyze` options and it does not require `cov-analyze` to be invoked with `--test-advisor` option. This option turns on all the necessary passes of the analysis that are required to compute test metrics.

Test metrics contain a mapping from the tests available in the emit directory to the functions that each test covers. This data is used during Test Prioritization to calculate properties of the tests (for example, the `covered_function_count` property) which can be used as part of the test score.

The `-enable-test-metrics-` option requires `--strip-path` option.

This option works with C/C++, C#, and Java.

--enable-virtual
Enables full, virtual-call resolution. By default, a C++ analysis treats all virtual functions as unimplemented, whereas full virtual call resolution is enabled by default for Java and C# analyses. When specified, this option allows up to 100 function resolutions for any virtual method. If that limit is exceeded, the analysis engine reverts to the default behavior.

Do not use this option if you specify the `--enable-single-virtual` option. The analysis can take significantly longer than the default or when the `--enable-single-virtual` option is enabled because the analysis engine looks at all implementations of virtual functions, which can result in more defect reports.

Specify this option, or `--enable-single-virtual`, to enable interprocedural analysis of Apple Block invocations in C and C++ code.

Note

To make the analysis resolve to a model of a virtual or pure virtual function without using `--enable-virtual`, see "Analyzing models of virtual functions" in the Coverity 8.0 Checker Reference.

--export-summaries <true|false>
Collects function summary data for the analysis. The collected data provides interprocedural analysis information for Desktop Analysis users, and must be committed to any stream that is used by Desktop Analysis.

This option is `true` by default.

--field-offset-escape
[C++ analysis option] A pointer escapes the analysis if it is written to memory, passed to `free()`, or passed to a function definition that is inaccessible to `cov-analyze`. Once the pointer escapes the analysis, the storage to which it points will never be treated as a leak or uninitialized.

This option eliminates certain false positives in C++ by making the analysis treat `&v->field` as an alias for `v` because some programs exploit the fact that `(&v->field) - offsetof(typeof(v), field) == v` to free `v` given `&v->field`.

By default, this heuristic applies to only to C code (but not C++). This option enables this heuristic for C++, as well.

See also, `--no-field-offset-escape`.

--force
Turns off incremental analysis. This setting forces a full re-analysis of the source, even if the source file or other source files on which it depends have not changed since it was previously analyzed.

--fnptr-models
[C/C++ analysis option] Enables function pointer models if the analysis fails to analyze certain function pointers calls. You can enable analysis of calls to function pointers, without requiring explicit models, using the --enable-fnptr option. For more information and examples, see "Modeling function pointers" in the Coverity 8.0 Checker Reference.

--handle-badalloc
[C/C++ analysis option] Causes the analysis as a whole to handle exceptions of type `std::bad_alloc`, both for exceptional control flow and for UNCAUGHT_EXCEPT. By default, such exceptions are otherwise ignored even when you use `--enable-exceptions`.

For an example, see --enable-exceptions.

--hfa
[C-only analysis option] Reports unnecessary header file includes. For more information, see "HFA" in the Coverity 8.0 Checker Reference.

The `-all` option does not enable this checker.

--hibernate-config
Specifies a directory that contains Hibernate mapping XML files, if applicable. Pertains to the HIBERNATE_BAD_HASHCODE checker (see the *Coverity 8.0 Checker Reference* for details).

--inherit-taint-from-unions
Enable taint to flow downwards from a C/C++ union to its component fields. This is required to check code that writes to a union using `memcpy(&u, &tainted, n)` and later reads using `u.field`.

Affects security checkers TAINTED_SCALAR and INTEGER_OVERFLOW.

--java
[Java analysis option] Restricts the analysis to Java code (even if code from other languages have been captured through a build to the same emit location in the intermediate directory). It is an error to use this option with the `--cpp` or `--cs` options.

--jobs <number-of-workers> | `auto` | `max`<number-of-workers>, -j <number-of-workers> | `auto` | `max`<number-of-workers>
Allows you to control the number of analysis workers that run in parallel, subject to any limits specified by your license. Starting in version 7.6.0, the need to use this option should be rare because the default typically sets the appropriate number of workers to use for your hardware, license, and analysis task. Note that the default for this option varies by license.

- Default for a non-Flexnet license (`license.dat`): `--jobs auto`
- Default for a Flexnet license: `--jobs max8`

In general, the analysis runs faster with more threads, but the scalability of that speed increase depends on the kind of analysis, the code language(s), and other properties of the code base. In general, analysis of C code parallelizes best, followed by C++, followed by C# and Java quality analyses, followed by Web Application Security analysis, which is largely not parallelized.

This option must specify one of the following values:

- `<number-of-workers>`: Specifies number of analysis workers to run in parallel.

  Example: `--jobs 8`

  The specified number of workers is not allowed to exceed suggested limits for your hardware unless you also use the `--override-worker-limit` option.

- `auto`: Automatically determines the number of workers to use. Hardware detection through `--jobs auto` attempts to optimize for the case where the analysis has full or nearly-full use of the machine's computational resources (memory and CPU). If that is not the case, you should consider setting `-j` explicitly, for example, where the analysis occupies one of several "executors" on a continuous integration server.

  Example: `--jobs auto`

  If `--jobs auto` is set, the analysis will determine the number of workers to run based on the minimum of the following:

  - The largest number of workers that keeps the recommended minimum physical memory requirements below the actual physical memory of the machine.
  - The number of logical CPUs, or virtual cores, on the machine. This is the number of threads or processes that the operating system can schedule simultaneously on the hardware.
  - Six (6) times the number of "physical" CPU cores, when known.
  - 48: Coverity has not found performance improvements from using more than 48 workers, and using more might reach limits on open file descriptors, and so on.
  - The number permitted by the license or available for Flexnet checkout.

  Detection of memory and logical CPUs should work on all analysis platforms, but detection might fail or produce incorrect results in some virtualization environments.

  This value is not compatible with the `--override-worker-limit` option.

- `max<number-of-workers>`: Limits the number of analysis workers that can run in parallel based on the maximum you set and the amount of memory and number of cores that are available.

  Example: `--jobs max8`

  This value is not compatible with the `--override-worker-limit` option.

See Running a Parallel Analysis and Running a Parallel Analysis with Coverity Analysis for Java for guidance.

For backward compatibility, the `--j` <number-of-workers> syntax is still supported in this release.

--jvm-max-mem
: [Java Web Application Security option] Sets the value of the heap size for the JVMs used by the Java Web Application Security dynamic analyzer and framework analyzer components, and for the

Findbugs analysis. The option specifies the size (an integer) in megabytes. That is, this options sets the options `--fb-max-mem`, `--da-max-mem` and `--framework-analyzer-max-mem`. The three options will override the heap size set by `--jvm-max-mem` for a particular JVM.

--list-checkers
Displays a list of checkers that are available in the current release. Each entry indicates whether the checker is enabled by default, and if not, how you can enable it. Some require the use of `--enable`, while others can be enabled with other options (for example, `--concurrency` or `--security`), as well. For detailed information about checkers, see the *Coverity 8.0 Checker Reference*.

--max-loop <num>
Limits the maximum number of times that loops are traversed. The default limit is 32, which should be encountered rarely. -1 means unlimited.

--max-mem <value>
Sets the maximum amount of memory, in megabytes, that a single analysis worker process will use for the core analysis. The total memory required is approximately the product of the `--max-mem` and `-j` options. The default value is 512.

The worker will use some additional memory for miscellaneous purposes, and even more memory if you use `--enable-constraint-fpp` or enable INTEGER_OVERFLOW.

The analysis will reject a setting that is too large for the available physical memory and number of workers.

On 32-bit Windows systems, do not set `<value>` to more than 512 megabytes.

out-of-memory error

An out-of-memory error indicates that the analysis is trying to use more memory than is available to the system. If an out-of-memory error occurs, use the `--max-mem` option to *decrease* the amount of memory that the analysis is allowed to use.

Note that JVM max-mem options work in the opposite way. So it is necessary to *increase* (not decrease) the max-mem for out-of-memory errors related to the JVM. In this case, it is possible that the system will run out of memory in the JVM.

--misra-config <path/to/misra_configuration_file>
[C/C++ MISRA option. Required only for a MISRA analysis.] Provides the path to a configuration file for a MISRA standard (required) and one of seven increasingly strict subsets of MISRA rules and directives within the standard. Level 7 is the most strict because it applies all supported rules and directives for a given standard. Level 1 is least strict. You can only pass one configuration file per MISRA analysis run. If you want analysis results for multiple MISRA standards, you must run separate analyses for each standard.

MISRA analyses can run together with non-MISRA analyses. To run a MISRA-only analysis, you can use the `--disable-default` option with `--misra-config`.

To use `--misra-config` with the `cov-run-desktop` command, you must also specify the `--whole-program` option.

A configuration file can specify only one of the following standards:

- `c2004` - MISRA C 2004
- `cpp2008` - MISRA C++ 2008
- `c2012` - MISRA C 2012

Recommendation

Coverity recommends that you use one of the configurations provided in `<install_dir>/config/MISRA`. There is one configuration file for each standard at level 1,level 2, level 3, level 4, level 5, level 6 and level 7. Each configuration file has a field called `deviations`. No violations will be reported for the rules specified in the `deviations` field. Note that the level 7 specification applies all supported rules and directives in the standard and is equivalent to specifying no level at all.

If you want to define your own level of compliance, you should create and edit a copy of the configuration file instead of editing the file that Coverity provides. Using the copy will prevent the loss of your configuration upon upgrade and avoid the potential for other undesired behavior. Coverity also recommends adding the copy to your source stream to ensure that the history of changes to that file are tracked.

For MISRA rules and directives, see "MISRA Rules and Directives" in the Coverity 8.0 Checker Reference.

For the MISRA analysis workflow, see "Running MISRA analyses" in Coverity Analysis 8.0 User and Administrator Guide.

--model-file <file>.xmldb
Uses the specified file to override any function models that are automatically derived from the implementation. It can determine whether a specified file is for user or derived models. Note that if the default file at `<install_dir_sa>/config/user_models.xmldb` exists, it is used even without specifying this option. This option can be specified multiple times.

You can use this option on the output of cov-collect-models or cov-make-library. For more information, see "Model search order" in the Coverity 8.0 Checker Reference.

--no-field-offset-escape
[C/C++ analysis option] Disables a heuristic that can cause RESOURCE_LEAK and UNINIT to produce false negatives when tracking aliases of pointers.

A pointer escapes the analysis if it is written to memory, passed to `free()`, or passed to a function whose definition is inaccessible to `cov-analyze`. Once the pointer escapes analysis, the storage to which it points will never be considered leaked or uninitialized.

To eliminate false positives in C code (but not C++ code), the analysis considers `&v->field` to be an alias for v because some programs exploit the fact that `(&v->field) - offsetof(typeof(v), field) == v` to free v given `&v->field`.

If a program does not use this idiom, this heuristic might lead to false negatives. For example, if you call `myfunction(&v->field)` when this heuristic is enabled, the analysis assumes that `v` escapes, so the analysis will not catch a RESOURCE_LEAK or UNINIT on `v`. This option disables the application o f that heuristic.

This option is set automatically if the `--aggressiveness-level` option is set to `medium` (or to `high`).

--no-java
Disables Java analysis. By default, the `cov-analyze` command otherwise analyses any Java code it finds in the intermediate directory.

--no-log
Disables logging.

--one-tu-per-psf <true|false>
Analyzes exactly one translation unit (TU) found for a given primary source file name. Coverity Analysis uses an algorithm to choose the appropriate TU to analyze. It analyzes the latest TU *except in some cases*, for example, where a single, parallel `cov-build` command was used and the latest compiled version of the file might change from build to build, causing non-determinism.. A `false` value enables analysis of all TUs, regardless of primary source file duplication. The default value is `true`.

--output-tag <name>
Specifies a non-default location within the intermediate directory for the results of one or more analyses. The name can be anything you choose, using characters allowed in file names. When specified *without* the `--append` option, prior results found in that location are replaced. When specified *with* `--append`, new results are added to the result set.

--override-worker-limit
Allows you to specify a value to `-j` that is greater than the recommended value. This option can be useful when the license allows more workers than the number of cores in the machine.

--parse-warnings-config <filename>
[C/C++ analysis option] Specifies the name for the configuration file, which allows you to change the parse warnings that pass through a warning filter. For a sample, see `config/parse_warnings.conf.sample`. See also `--enable-parse-warnings`.

--path-log-threshold <number>
If a function has more than <number> paths, this count is output to the log file.

--paths <number>
Sets the upper limit on the number of paths to traverse for each function. Default is 5000.

--preview
Enables almost all preview checkers. Exceptions are noted below.

Preview checkers have not been validated for regular production use. Results from this checker could have higher false positives and false negatives compared to standard checkers. Also, changes to this checker in the next release could change the number of issues found. Preview checkers are included

in this release so that you can use the checker in a test environment and evaluate its results. Please provide feedback to `support@coverity.com` on its accuracy and value.

Exceptions
The `--preview` option *does not* enable the following preview checkers:

- All concurrency, rule, security, or Symbian checkers (some of which are classified as preview checkers). To enable such checkers, you use the respective `--concurrency`, `--rule`, `--security`, or `--symbian` option.
- Preview checkers that perform Web application security analyses. To enable them, see --webapp-security-preview and --webapp-config-checkers.
- ENUM_AS_BOOLEAN
- MISRA_CAST

To enable these checkers, you can use the --enable option.

For a list of preview checkers, you can use list-checkers.

--print-paths
Prints the number of paths explored for each analyzed function.

--rule
[C/C++ analysis option] Enables rule checkers.

For a list of rule checkers, you can use list-checkers.

--security
[C/C++ analysis option] Enables C/C++ security-related checkers that are not preview checkers.

For a list of the security checkers to which this option applies, you can use --list-checkers.

--security-file <`license file`>, -sf <`license file`>
Path to a valid Coverity Analysis license file. If not specified, this path is given by the <`security_file`> tag in the Coverity configuration or by `license.dat` (located in the Coverity Analysis <`install_dir`>`/bin`). A valid license file is required to run the analysis.

--strip-path <path>, -s <path>
Strips the prefix of a file name path in error messages and references to your source files. If you specify the `--strip-path` option multiple times, you strip all of the prefixes from the file names, in the order in which you specify the `--strip-path` argument values.

This option is also available with `cov-commit-defects` and `cov-import-results`.

The leading portion of the path is omitted if it matches a value specified by the this option. For example, if the actual full pathname of a file is `/foo/bar/baz.c`, and `--strip-path /foo` is specified, then the name attribute for the file becomes `/bar/baz.c`.

Important!

Coverity recommends using this option for a number of reasons:

Failure to use this option can result in poor Coverity Connect performance, triage issues related to component maps, an unnecessary increase the size of the Coverity Connect database, and even incorrect LOC counts.

This option shortens paths that Coverity Connect displays. It also allows your deployment to be more portable if you need to move it to a new machine in the future.

In addition, using this option during the analysis, rather than when committing the analysis results to Coverity Connect, can enhance end-to-end performance of the path stripping process itself.

The `--strip-path option` is mandatory when running `cov-analyze --test-advisor`. This changes how file-level violations are merged with existing violations in Coverity Connect. Violations generated with Test Advisor 8.0 will not be merged with violations generated with previous versions of Test Advisor, and existing triage of previously-generated Test Advisor violations will not be associated with the new violations.

Linux example:

```
> cov-analyze --dir myDir --strip-path=`pwd`
```

Windows example:

```
> cov-analyze --dir myDir --strip-path=%cd%
```

In the less common case, the option should specify the root of your build tree.

--symbian
[C++ analysis option] Enables all Symbian checkers.

For a list of Symbian checkers, you can use list-checkers.

--ticker-mode <mode>
Sets the mode of the progress bar ticker display to:

none
No progress bar at all.

no-spin
Print stars only, without the spinning bar.

spin
(default) Stars with a spinning bar at the end. Each analyzed function corresponds to one step of spin.

--tu <translation_unit_id(s)>, -tu <translation_unit_id(s)>
Limits the scope of `cov-analyze` to a set of translation units (TUs), named by their numeric ID attribute(s). A translation unit approximately maps to the output from a single run of a compiler. This option requires a comma-separated list of id(s), and `--tu` can be specified multiple times. The union of all these identifier sets is the set of TUs to operate on subsequently, for operations that work on

TUs. It is an error if any of the specified IDs do not correspond to any existing translation unit. To get the IDs for translation units, use the `cov-manage-emit` list sub-command.

You can use the `--tu` and `--tu-pattern` options together.

--tu-pattern <translation_unit_pattern>, -tp <translation_unit_pattern>
Limits the scope of `cov-analyze` to a set of translation units specified with a translation unit pattern. The `--tu-pattern` option can be specified multiple times. Matching TU sets are unioned together across all patterns.

Both `--tu` and `--tu-pattern` can be specified on a single command line. The final set of TUs operated upon includes a given TU if it matches any specified translation unit pattern or its ID is listed explicitly as an argument to `--tu`.

It is an error if at least one `--tu-pattern` argument is specified but no translation unit matches any of the specified patterns.

You can get useful information on TUs by using the `cov-manage-emit` list sub-command.

See Translation unit matching for more information.

--use-jshintrc <path/to/your/.jshintrc>
[JavaScript analysis] Identifies a custom configuration to use for the JSHint analysis in place of the default Coverity configuration file (see http://jshint.com/docs/ for information about `.jshintrc` file configuration).

Note that hierarchical configuration files, based on the source code directory hierarchy, are not supported, due to limitations in capture and in how `cov-analyze` invokes JSHint.

When using `--use-jshintrc`, you must also pass --enable-jshint.

--user-model-file <user_file.xmldb>
[Deprecated as of version 7.7.0] This option will be removed and replaced in a future release. Use --model-file instead.

--wait-for-license
Indicates that if a license cannot be obtained from the license server, `cov-analyze` must wait until a license becomes available. After a license becomes available, `cov-analyze` acquires it and proceeds with the analysis. This option is ignored if `cov-analyze` does not use a floating-node license.

Extend SDK options

--dtd <directory>
[Deprecated Extend SDK analysis option] Use the --prevent-root option instead.

--prevent-root
[Extend SDK analysis option] When running a Extend SDK checker, specify the location of the Coverity Analysis installation directory:

```
--prevent-root /<install_dir_sa>
```

See *Coverity Extend SDK 8.0 Checker Development Guide* for more information.

Java FindBugs options

--disable-fb
[FindBugs analysis option for Java] Disables FindBugs analysis.

An error will occur if you combine `--enable-fb` with `--disable-fb`.

--enable-fb
[FindBugs analysis option for Java] Enables FindBugs (version 2.0.1) analysis (requires a Quality Advisor license). You can use this option to enable FindBugs analysis while disabling all other default checkers through the `--disable-default` option. If you attempt to run an analysis when using this option, but you do not have a Quality Advisor license, Coverity Analysis will disable the checkers and print an error message.

FindBugs is an open source program for finding bugs (defects) in Java code. It provides a large group of FindBugs bug patterns that can detect a wide variety of defects.

A FindBugs analysis complements the analysis with Coverity Analysis for Java checkers. By default, `cov-analyze` detects those medium-priority and high-priority FindBugs defects that do not generate too many unimportant results (for details, see `<install_dir_sa>/findbugs-ext/config/include.coverity-default.xml`). The default analysis also excludes results that overlap with non-deprecated Coverity Analysis for Java checkers. However, you can include or exclude other bugs from the analysis. To refine the FindBugs results, see `--fb-include`, `--fb-exclude`, and `--disable`.

You can use `cov-analyze` to run a FindBugs analysis on builds in your intermediate directory, which are generated by `cov-build` or by `cov-emit-java`. However, Coverity Analysis for Java does not integrate with FindBugs running outside of the `cov-analyze` command. The FindBugs analysis should add no more than about 20% to the running time of `cov-analyze`, assuming that most Coverity checkers are enabled.

Coverity Analysis uses the FB prefix to distinguish defects that match FindBugs bug patterns from defects found by Coverity checkers. For example, Coverity Analysis for Java uses FB.DM_EXIT for the FindBugs DM_EXIT bug pattern. The console prints a summary of the results, which looks something like the following:

```
FindBugs Checkers: 74 errors
FB.DE_MIGHT_IGNORE                                1
FB.DM_EXIT                                        5
FB.DP_CREATE_CLASSLOADER_INSIDE_DO_PRIVILEGED     2
FB.EI_EXPOSE_REP                                 14
...
```

Keep in mind that the defect summary identifies the number of defect occurrences, which is likely to be somewhat larger than the number of CIDs in Coverity Connect.

When you run the analysis, the console output looks something like the following:

```
[STATUS] Running FindBugs:
Scanning archives (2 / 2)
2 analysis passes to perform
Pass 1: Analyzing classes (2862 / 2862) - 100% complete
Pass 2: Analyzing classes (1793 / 1793) - 100% complete
Done with analysis
```

An error will occur if you combine `--disable-fb` with `--enable-fb`.

After running the analysis, you continue to run `cov-commit-defects` to commit the analysis results to Coverity Connect. Because the FindBugs bugs are Static Java defects, Coverity Connect displays them in a Static Java stream along with the results of the Coverity Analysis for Java checkers. Coverity Connect lists defects found by FindBugs according to the bug categories that come from FindBugs, such as "FindBugs: Correctness."

--fb-dont-exclude-overlap
[FindBugs analysis option for Java. *Not recommended for production use*] Disables the default exclusion of certain FindBugs checkers. By default, `cov-analyze` prevents FindBugs analysis from reporting defects that are best found using Coverity checkers. In other words, Coverity checkers are better at finding some portion of the defects FindBugs can report because the defects reported by the Coverity checkers generally have lower false positive (FP) rates and consist of more true positives. This "overlap" with FindBugs is excluded by default when `cov-analyze` invokes FindBugs analysis.

After using this option to disable the default exclusions, you can then customize the exclusions with `--fb-exclude option` and a filter file. You can base your file filter on `exclude.overlap.xml` filter file.

--fb-exclude concurrency | <filter_file_name>
[FindBugs analysis option for Java] Specifies a set of FindBugs bug patterns to exclude when you run an analysis. By default, the analysis excludes FindBugs results that overlap with non-deprecated Coverity Analysis for Java checkers because the latter return more true positive defects, fewer false positive defects, or both in cases where such overlap occurs. You can use `--fb-exclude` to exclude additional defects from the analysis results. This option accepts one or more of the following values:

- `concurrency`

  Exclude concurrency-related FindBugs results. For details, see `<install_dir>/findbugs-ext/config/exclude.concurrency.xml`.

- `<filter_file_name>`

  Uses a FindBugs filter file to specify the FindBugs bug patterns or bug categories that you want to exclude. For example, to produce results equivalent to `--disable FB.DE_MIGHT_DROP --disable FB.DE_MIGHT_IGNORE`, you can specify the following in your filter file:

```
<FindBugsFilter>
  <Match>
    <Bug pattern="DE_MIGHT_DROP,DE_MIGHT_IGNORE"/>
```

```
    </Match>
  </FindBugsFilter>
```

To create a filter file, see the FindBugs documentation at http://findbugs.sourceforge.net/manual/filter.html. Note that you can use filter files not only to exclude or include individual bug patterns, but also to exclude or include categories of FindBugs bugs. For example, to exclude medium-priority and low-priority bugs in the FindBugs MT_CORRECTNESS (Multi-threaded correctness) category, you can use the following:

```
<FindBugsFilter>
  <Match>
    <Bug category="MT_CORRECTNESS"/>
    <Or>
      <Priority value="2"/>
      <Priority value="3"/>
    </Or>
  </Match>
</FindBugsFilter>
```

You can specify filter files using relative or absolute paths. If you are using Cygwin, specify Windows native paths, not Cygwin paths.

You can specify this option multiple times on the command line. The order of the exclusions does not matter.

You can also use the `--fb-include` option. If a defect is excluded, it will not appear in the results even if the defect has been included through `--fb-include`.

--fb-include high-priority | medium-priority | low-priority | <filter_file_name>
[FindBugs analysis option for Java] Specifies a set of FindBugs bug patterns to run on your code. You can use this option to find defects based on FindBugs priority categories, or you can use a filter file for this purpose.

This option accepts only a single value. For example:

- `--fb-include low-priority`

  Returns the most FindBugs results.

- `--fb-include medium-priority`

  Returns only the defects that FindBugs identifies as medium-priority or high-priority bugs.

  Note that `cov-analyze` detects a significant subset of these defects by default (for details, see `<install_dir_sa>/findbugs-ext/config/include.coverity-default.xml`).

- `--fb-include high-priority`

  Returns only the defects that FindBugs identifies as high-priority bugs.

- `--fb-include my_inclusions.xml`

Uses a FindBugs filter file to specify the FindBugs bug patterns or bug categories that you want to include. For guidance, see the `--fb-exclude` option for filter files.

FindBugs rates bugs based on their priority (high, medium, or low). These priorities are similar to the Coverity defect impacts described in each Coverity Integrity Report. In FindBugs, high-priority defects are most likely to be actionable, while low-priority defects are least likely to require action. Note that Coverity Integrity Report rates bugs in the following FindBugs categories as medium-impact defects: Correctness, Multithreaded correctness, and Security. All other categories of bugs are rated as low-impact defects.

You can also use the `--fb-exclude` or `--disable` option. If a defect is excluded or disabled, it will not appear in the results even if the defect has been included through `--fb-include`.

--fb-max-mem
[FindBugs analysis option for Java] Sets the JVM heap size of the VM that is running FindBugs. This option is similar to `--max-mem` in that it takes an integral value of megabytes, but differs in that the action to take if FindBugs execution runs out of memory is to provide a larger value. If the option is not specified, the default value is 1024.

Web Application Security options

--add-password-regex <regular_expression>
[Web Application Security option] Treats field and method parameter names that match the specified regular expression as a password source. You can specify this option multiple times. Note that if you use the `--add-password-regex` and `--replace-password-regex`, the default regular expression will be replaced, then extended.

This option affects analysis by the WEAK_PASSWORD_HASH and SENSITIVE_DATA_LEAK checkers. See also, --replace-password-regex.

--allow-jsp-include-param-blacklist
[Java Web Application Security option] Treats any servlet request parameters that are set through a `<jsp:include>` tag as untainted. This option reduces false positives when a servlet request parameter that is used from an included JSP file never contains tainted data but increases the risk of false negatives in cases where the parameter can be tainted.

This setting changes the default behavior of the XSS checker, a Web application security checker.

--disable-webapp-config-checkers
[Web Application Security option] Disables Web application configuration checkers, the Java CONFIG.* checkers (for example, CONFIG.DUPLICATE_SERVLET_DEFINITION).

Note that `--webapp-config-checkers` and `--disable-webapp-security` also apply to Java Web application configuration checkers.

--disable-webapp-security
[Web Application Security option] Disables the Web application security checkers. Note that these checkers are disabled by default.

See `--webapp-security`.

--distrust-all
[Web Application Security option] This option is equivalent to setting all the `--distrust-*` options. It applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 8.0 Checker Reference*.

This option cannot be used with --trust-all.

--distrust-console
[Web Application Security option] Treats data obtained from a console (for example, reading from `System.in`) as though it is tainted. Such data is otherwise trusted by default. This option applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 8.0 Checker Reference*.

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following example produces an issue report:

```
           public class ConsoleInj {
             public void testInjection(Statement stmt) throws Exception {
               BufferedReader reader = new BufferedReader(new
InputStreamReader(System.in));
               String query = reader.readLine();
               stmt.executeQuery(query);
             }
           }
```

This option cannot be used with --trust-console.

--distrust-cookie
[Web Application Security option] Specifies the default behavior of the analysis, which is to distrust data from HTTP cookies and treat it as though it is tainted. This option applies to all the checkers of type Security (Tainted dataflow checker). See the *Coverity 8.0 Checker Reference* for details.

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following example produces an issue report:

```
           class SqlInjFromCookie extends HttpServlet {
               Statement sql_stmt;
               public void doPost(HttpServletRequest req, HttpServletResponse
resp) {
                   try {
                       sql_stmt.executeQuery(req.getCookies()[0].getValue());
                   } catch(Exception e) {
                       // ...
                   }
               }
```

```
            }
```

This option cannot be used with --trust-cookie

--distrust-database
Treats data obtained from a database (for example, SQL query results and Hibernate objects) as though it is tainted. Such data is otherwise trusted by default. This option applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 8.0 Checker Reference*.

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following example produces an issue report:

```
           public class DatabaseInj {
             public void testInjection(int columnIndex, Statement stmt) throws
Exception {
               ResultSet rs = stmt.executeQuery("SELECT * FROM *");
               String query = "SELECT * FROM " + rs.getString(columnIndex);
               stmt.executeQuery(query);
             }
           }
```

This option cannot be used with --trust-database.

--distrust-environment
[Web Application Security option] Treats data that the checker identifies as environment variables as though it is tainted. Such data is otherwise trusted by default. This option applies to all the checkers in the group Security (Tainted data checker). For details, see the *Coverity 8.0 Checker Reference*.

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following example produces an issue report:

```
       public class EnvironmentInj {
         public void testInjection(Statement stmt, String getVar) throws Exception
{
           String envVar = System.getEnv(getVar);
           String query = "SELECT * FROM " + envVar;
           stmt.executeQuery(query);
         }
       }
```

This option cannot be used with --trust-environment.

--distrust-filesystem
[Web Application Security option] Treats data obtained from a file system as though it is tainted. Such data is otherwise trusted by default. This option applies to all the checkers of type Security (Tainted dataflow checker). For details, see the *Coverity 8.0 Checker Reference*.

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

`--trust-filesystem`.

The following example produces an issue report:

```
            public class FilesysInj {
              public void testRead(FileInputStream fis) throws Exception {
                byte[] b = new byte[50];
                fis.read(b);
                stmt.executeQuery("SELECT * FROM " + new String(b));
              }
            }
```

This option cannot be used with --trust-filesystem.

--distrust-http
[Web Application Security option] Specifies the default behavior of the analysis, which is to treat Web input (for example, `GET` and `POST` parameters, and cookies) as though it is tainted. This option applies to all the checkers of type Security (Tainted dataflow checker). For details, see the *Coverity 8.0 Checker Reference*.

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following Java example produces an issue report:

```
            class ServletInj extends HttpServlet {
              Statement sql_stmt;
              public void doPost(HttpServletRequest req, HttpServletResponse resp)
{
                try {
                   sql_stmt.executeQuery(req.getParameter("x"));
                 } catch(Exception e) {
                   // ...
                 }
               }
            }
```

This option cannot be used with --trust-http.

--distrust-http-header
[Web Application Security option] Specifies the default behavior of the analysis, which is to distrust data from HTTP headers as though it is tainted. This option applies to all the checkers of type *Security (Tainted dataflow checker)*. For details, see the *Coverity 8.0 Checker Reference*.

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following example produces an issue report:

```
            class HttpHeaderInj extends HttpServlet {
              Statement sql_stmt;
              public void doPost(HttpServletRequest req, HttpServletResponse resp)
{
                try {
                  sql_stmt.executeQuery(req.getHeader("user-agent"));
                } catch(Exception e) {
                   // ...
                }
              }
            }
```

This option cannot be used with --trust-http-header.

--distrust-network
[Web Application Security option] Specifies the default behavior of the analysis, which is to treat data obtained from a network connection (for example, a TCP socket or HTTP connection) as though it is tainted. This option applies to all the checkers of type Security (Tainted dataflow checker). For details, see the *Coverity 8.0 Checker Reference*.

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following example produces an issue report:

```
            class NetworkInj {
              public void func(Socket s, Statement stmt) throws SQLException,
IOException {
                InputStream is = s.getInputStream();
                InputStreamReader isr = new InputStreamReader(is);
                BufferedReader br = new BufferedReader(isr);
                String query = br.readLine();

                query = "SELECT * FROM " + query;
                stmt.executeQuery(query);
              }
            }
```

This option cannot be used with --trust-network.

--distrust-rpc
[Web Application Security option] Specifies the default behavior of the analysis, which is to distrust data obtained from a Remote Procedure Call (RPC) as though it is tainted. This option applies to all the checkers in the group Security (Tainted data checker). For details, see the *Coverity 8.0 Checker Reference*.

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following example, which uses an Enterprise Java Bean (EJB), produces an issue report:

```
@Remote(RemoteInterface.class)
public class TestEJB implements RemoteInterface {
  Statement stmt;
  public void testWrite(String taint) {
    ResultSet rs = stmt.executeQuery("SELECT * FROM *");
    String query = "SELECT * FROM " + rs.getString(columnIndex);
    stmt.executeQuery(query);
  }
}
```

This option cannot be used with --trust-rpc.

--distrust-servlet
[Deprecated Web Application Security option] This option has been deprecated as of version 7.7.0 and will be removed from a future release. Use --distrust-http, instead.

This option cannot be used with --trust-servlet.

--distrust-system-properties
[Web Application Security option] Treats system properties (those obtained from `System.getProperty()`) as though they are tainted. Such properties are otherwise trusted by default. This option applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 8.0 Checker Reference*.

Note that the checker-level version of this option (not available to all checkers in this group) overrides the command-level version.

The following example produces an issue report:

```
public class SystemPropertiesInj {
   public void testInjection(Statement stmt, String p) throws Exception {
     stmt.executeQuery("SELECT * FROM " + System.getProperty(p));
   }
 }
```

This option cannot be used with --trust-system-properties.

--framework-analyzer-max-mem
[Web Application Security option] Increases the maximum on memory usage (specified in megabytes) by the framework analyzer. The default value is `1536`.

If the JVM that is running the framework analyzer runs out of memory, use this option to increase the amount of memory available to that JVM. Note that default value is high, so it is unlikely that you will need to increase it.

This option is analogous to the `--fb-max-mem` option for the FindBugs JVM.

--framework-analyzer-timeout
[Web Application Security option] Increase the timeout (specified in minutes) for the framework analyzer. The default value is `60` (60 minutes).

Use this option if the framework analyzer takes too long and is killed. Note that the need to use this option suggests that the hardware in use is most likely overloaded and not powerful enough for the analysis.

--not-tainted-field <fully_qualified_field_name>
[Web Application Security option] The value `<fully_qualified_field_name>` is a Perl regular expression describing a fully qualified field name. Any matching fields will be asserted to be untainted. Additional defects may be reported by the TAINT_ASSERT checker, but reported issues involving unsafe uses of the value will be suppressed in the Web application security checkers.

The option can be specified multiple times on a single command line.

See Adding Assertions that Fields are Tainted or Not Tainted for details.

--replace-password-regex <regular_expression>
[Web Application Security option] Replaces the default regular expression that the checker uses to infer passwords. You can specify this option only once. Note that if you use the `--add-password-regex` and `--replace-password-regex`, the default regular expression will be replaced, then extended.

This option affects analysis by the WEAK_PASSWORD_HASH and SENSITIVE_DATA_LEAK checkers. See also, --add-password-regex.

--skip-webapp-sanity-check
[Java-only Web Application Security option] Suppresses the warning message that normally appears if any Web application security checkers are enabled but `cov-emit-java --webapp-archive` was not used to emit the Web application (web-app) archive or directory.

The check, which this option overrides, is designed to catch the case where someone intended to run Web Application Security checkers but forgot to emit the WAR file. It is technically possible, but highly unlikely, for Java classes to contain an entire Web application (without any JSPs or framework configuration), in which case there would be no need for a WAR file.

For additional details, see --webapp-security and --skip-war-sanity-check.

--tainted-field <fully_qualified_field_name>
[Web Application Security option] Takes a Perl-style regular expression that describes a fully qualified field name. Any matching fields will be asserted to be tainted. Additional defects may be reported by the Web Application Security checkers, if any of the specified fields are used in an unsafe manner. The option can be specified multiple times on a single command line. As an example, passing the command line option `–tainted-field com.coverity.examples.Table.*` will assert that the fields title and values are tainted in the following code.

```
Package com.coverity.examples;

class Table {
  String title;
  String value;
  int id;
}
```

See Adding Assertions that Fields are Tainted or Not Tainted for more information.

--trust-all
[Web Application Security option] This option is equivalent to providing all the `--trust-*` options. This option applies to all the checkers in the group Security (Tainted data checker). For details, see the *Coverity 8.0 Checker Reference*.

This option cannot be used with --distrust-all.

--trust-console
[Web Application Security option] Specifies the default behavior of the analysis, which is to treat data obtained from a console (for example, reading from `System.in`) as though it is not tainted. This option applies to all the checkers in the group Security (Tainted data checker). For details, see the *Coverity 8.0 Checker Reference*.

This option cannot be used with --distrust-console.

--trust-cookie
[Web Application Security option] Treats data that is obtained from an HTTP cookie as though it is not tainted. Such data is otherwise distrusted by default. This option applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 8.0 Checker Reference*.

This option cannot be used with --distrust-cookie

--trust-database
[Web Application Security option] Specifies the default behavior of the analysis, which is to treat data obtained from a database (for example, SQL query results and Hibernate objects) as though it is not tainted. This option applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 8.0 Checker Reference*.

This option cannot be used with --distrust-database.

--trust-environment
[Web Application Security option] Specifies the default behavior of the analysis, which is to treat data from environment variables as though it is not tainted. This option applies to all the checkers in the group Security (Tainted data checker). For details, see the *Coverity 8.0 Checker Reference*.

Note that the analysis trusts data from environment variables by default.

This option cannot be used with --distrust-environment.

--trust-filesystem
[Web Application Security option] Specifies the default behavior of the analysis, which is to treat data obtained from a file system as though it is not tainted. This option applies to all the checkers in the group Security (Tainted data checker). For details, see the *Coverity 8.0 Checker Reference*.

Note that the analysis trusts data from filesystem sources by default.

This option cannot be used with --distrust-filesystem.

--trust-http
[Web Application Security option] Treats Web input (for example, `GET` and `POST` parameters, and cookies) as though it is not tainted. Web input is otherwise treated as tainted by default. This option applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 8.0 Checker Reference*.

This option cannot be used with --distrust-http.

--trust-http-header
[Web Application Security option] Treats data that is obtained from an HTTP header as though it is not tainted. Such data is otherwise distrusted by default. This option applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 8.0 Checker Reference*.

This option cannot be used with --distrust-http-header.

--trust-network
[Web Application Security option] Treats data obtained from a network connection (for example, a TCP socket or HTTP connection) as though it is not tainted. Such data is otherwise distrusted by default. This option applies to all the checkers in the group Security (Tainted dataflow checker). For details, see the *Coverity 8.0 Checker Reference*.

This option cannot be used with --distrust-network.

--trust-rpc
[Web Application Security option] Treats data obtained from a Remote Procedure Call (RPC) as though it is not tainted. Such data is otherwise distrusted by default. This option applies to all the checkers in the group Security (Tainted data checker). For details, see the *Coverity 8.0 Checker Reference*.

This option cannot be used with --distrust-rpc.

--trust-servlet
[Web Application Security option] This option has been deprecated as of version 7.7.0 and will be removed from a future release. Use with --trust-http, instead.

This option cannot be used with `--distrust-servlet`.

--trust-system-properties
[Web Application Security option] Specifies the default behavior of the analysis, which is to treat data obtained from system properties (for example, `System.getProperty()`) as though it is not tainted. This option applies to all the checkers in the group Security (Tainted data checker). For details, see the *Coverity 8.0 Checker Reference*.

This option cannot be used with --distrust-system-properties.

--webapp-config-checkers
[Web Application Security option] Enables Web application configuration checkers, the Java CONFIG.* checkers (for example, CONFIG.DUPLICATE_SERVLET_DEFINITION).

Note that `--disable-webapp-config-checkers` and `--disable-webapp-security` also apply to Java Web application configuration checkers.

--webapp-security
[Web Application Security option] Enables the production checkers that are used for Web Application Security analyses. To use these checkers, your Coverity license must include Coverity Security Advisor. If it does not include Coverity Security Advisor, you will receive an error if you attempt to run an analysis with security checkers enabled.

Java Prerequisite: Prior use of the `--webapp-archive` option to `cov-emit-java` for the WAR file. See Running a Security Analysis on a Java Web Application for details.

C# Recommendation: Coverity highly recommends using `cov-build` to capture an invocation of `Aspnet_compiler.exe` on your ASP.NET project root. See Running a Security Analysis on a Java Web Application for details.

Note

Use these checkers only if you need them because the security analysis adds non-trivial time and memory requirements to the overall analysis.

See also --webapp-security-preview.

--webapp-security-aggressiveness-level <low|medium|high>
[Web Application Security option] Tunes the aggressiveness of assumptions that the analysis makes to find potential security vulnerabilities (security defects). Higher levels report more defects, but the analysis time increases and memory usage is likely to increase. Higher levels also increase the likelihood that any given defect is a false positive. Values for level are low, medium, or high. Default is low.

This option can assist security auditors who need to see more defects than developers might need to see.

When analyzing code that uses unsupported Web application frameworks, medium or high aggressiveness levels can be more useful than the default.

--webapp-security-config <JSON_file>
[Java Web Application Security option] Takes a path to a JSON file with a number of user configuration directives. To create this file, see the reference documentation ("Appendix A. Java Web Application Security configuration file reference" in *Coverity 8.0 Checker Reference*).

--webapp-security-preview
[Web Application Security option] Enables preview checkers that are used for Web Application Security analyses (for example, PATH_MANIPULATION and HARDCODED_CREDENTIALS).

See also --webapp-security.

Test Advisor options

--compute-test-priority
[Test Advisor option] For Test Advisor test prioritization, this option creates a prioritized list of test (outputs to the location specified in `--test-priority-output`) using the specified test priority policy (specified in `--test-priority-policy`.

For usage information, see *Coverity Test Advisor and Test Prioritization 8.0 User and Administrator Guide*.

This option works with C/C++, C#, and Java.

--disable-test-metrics
Disables Test Advisor test prioritization metric data. If specified, this option takes precedence over `--enable-test-metrics`.

--test-advisor
Enables Test Advisor analysis. The --strip-path option is required for Test Advisor.

--test-advisor-policy <policy_file>
Specifies that the file is read as the policy for the Test Advisor run. The file must be a JSON file. For more information about construction policy files, see "Creating Test Advisor policies" in the *Coverity Test Advisor and Test Prioritization 8.0 User and Administrator Guide*.

--test-advisor-eval-output <eval_output_dir>
Causes the filter evaluation output to be written to the specified directory when Test Advisor runs. The specified directory is created if it does not already exist. All source files that are analyzed by Test Advisor are copied to this directory, and lines within each file that are excluded from analysis by the test policy are annotated with the filter that caused the exclusion.

--test-advisor-eval-rule <rule_number>
Causes the filter evaluation to be written only for the specified rule of the policy. Rules are numbered starting from `1` in order of their appearance in the policy file. This option has no effect unless `--test-advisor-eval-output` option is also used.

--test-advisor-verbose
Causes the output of verbose messages during the execution Test Advisor.

--test-priority-eval-output <eval_output_file>
For Test Advisor test prioritization, this option specifies the path and name for a file to which details about the analysis performed by test prioritization will be written. This option can be used to help debug the test prioritization policy.

The output file is in CSV format, where each line contains a record with the following information:

- A function, identified by its mangled name and filename
- A test, identified by its suitename and testname
- A rule from the policy file, identified by its name

Each record represents a function which passed the filters for the given rule, and a test which covers the function. If more than one test covers the function, that function shall have multiple records in the eval output, one for each test.

The `filtered_function_count` rule property for a given (`test,rule`) pair is thus the number of records which have that (`test,rule`) pair in the eval output. The functions listed in those records are the functions which contributed to the `filtered_function_count`. The eval output provides a way to validate that the `filtered_function_count` is correct. If a score in the test prioritization

output is not as expected, the functions contributing to the rule's `filtered_function_count` can be examined to determine why the discrepancy exists.

The first line of the eval output contains a header which indicates the format of the subsequent lines, and does not contain a record itself.

Due to the volume of data involved, the output file size may be large. The `--tu-pattern` option of `cov-analyze` can be used to restrict analysis to specific translation units in order to limit the size of the output.

--test-priority-output <output_file>
For Test Advisor test prioritization, this option specifies the path and name for the file to which the test prioritization output will be written.

For information about the output format, see "Test scoring policy language" in the *Coverity Test Advisor and Test Prioritization 8.0 User and Administrator Guide*.

--test-priority-policy <policy_file>
For Test Advisor test prioritization, this option specifies the path and name of the policy file which is used by test prioritization to determine how tests are scored. The file must be a JSON file.

For information about the policy file format, see "Test scoring policy language" in the *Coverity Test Advisor and Test Prioritization 8.0 User and Administrator Guide*.

Shared options

--config `<coverity_config.xml>` , -c `<coverity_config.xml>`
Use the specified configuration file instead of the default configuration file located at `<install_dir_sa>/config/coverity_config.xml`.

--debug, -g
Turn on basic debugging output.

--ident
Display the version of Coverity Analysis and build number.

--info
Display certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

--redirect stdout|stderr,<filename> -rd stdout|stderr,<filename>
Redirect either stdout or stderr to <filename>.

--tmpdir <tmp>, -t <tmp>
Specify the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>
Sets the detail level of command messages. Higher is more verbose (more messages). Defaults to 1. Use `--verbose 0` to disable progress bars.

Exit codes

- 0: The analysis was successful. Results should be considered usable and are ready to be committed with `cov-commit-defects`.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

Although the console output can provide diagnostics and warnings that might help to improve the analysis configuration, or suggest reporting "recoverable errors" to Coverity support, this information is auxiliary to the exit code. Users and scripts should rely on the exit code when determining whether to proceed in consuming the analysis results.

Examples

Analyze the intermediate directory at `/home/user/apache` using only the DEADCODE checker:

```
> cov-analyze --dir /home/user/apache --disable-default --enable DEADCODE
```

See Also

cov-build

cov-commit-defects

cov-make-library

Name

cov-analyze-java Note: Documentation merged with cov-analyze documentation.

Synopsis

`cov-analyze-java` --dir <intermediate_directory> [OPTIONS]

Description

This command is deprecated as of version 7.7.0 and will be removed from a future release. The documentation for this command has been merged with the cov-analyze command documentation.

See Also

cov-analyze

cov-build

cov-commit-defects

cov-emit-java

Name

cov-blame Compute Coverity Connect automatic owner assignment based on SCM history.

Synopsis

```
cov-blame
  --dir <intermediate_directory>
  --preview-report <filename>
  [--scm <scm_type>]
  [--scm-tool <scm_tool_path>]
  [--scm-project-root <scm_root_path>]
  [--scm-tool-arg <scm_tool_arg>]
  [--scm-command-arg <scm_command_arg>]
  [--no-triage-filters]
  [--owner-assignment-rules <RULE1>,<RULE2>...,<RULEn>]
  [--stop-after <limit>]
  [OPTIONS]
```

Description

The `cov-blame` command computes the automatic ownership assignments based on SCM (source code management) history and owner assignment rules. The SCM history consists of when, where, and by whom the code was changed and the ownership rules derive the owner of an issue based on the SCM history.

This command requires that source files remain their usual locations in the checked-out source tree. If the files are copied to a new location after checkout, the SCM query will not work.

There are two main use cases for this command:

1. **`cov-blame` is automatically called for owner assignment as part of the commit process.**

   In this case, `cov-commit-defects` automatically calls `cov-blame` based on owner assignment rules provided to the Coverity Connect UI. `cov-commit-defects` invokes `cov-blame` to compute the ownership assignments. If the rule assigned to the stream requires SCM data, `cov-blame` first attempts to retrieve SCM data for files related to defects from the intermediate directory. If SCM data has not been imported to the intermediate directory, the command can directly query the SCM for the relevant history data using the `--scm*` options defined in `cov-commit-defects`. The assigned owners (SCM users) are then written back into the intermediate directory. `cov-commit-defects` then picks up the ownership assignment files and Coverity Connect sets the owner accordingly in the triage pane.

2. **`cov-blame` is manually invoked to test and compare ownership rules.**

   You can invoke `cov-blame` to produce a report of owner assignments for defects to help you accomplish the following:

   - Verify, before you commit, that the automatic ownership is producing the proper/expected assignments. You can specify one or more owner assignment rules through the `--owner-assignment-rules` option for comparison.

- To compare automatic owner assignment to the owners that are already defined in Coverity Connect. In this way, you can see how successful an owner assignment rule would have been for historical defects that have already been manually assigned in Coverity Connect.

Before you run `cov-blame`, you must have a have an existing preview report or generate one using `cov-commit-defects --preview-report-v2`.

Options

--dir <int_dir>
: Pathname to an intermediate directory that is used to store the results of the build and analysis. This is required.

--no-triage-filters
: Calculate owner assignment for all defects whose merge key is present in the preview report, regardless of the triage values. By default, owner assignment is only calculated for defects whose current triage values satisfy the following conditions:

  - The Owner attribute is unset
  - The Classification attribute is Unclassified, Pending, Bug, or Untested.

  This option is useful, for example, to calculate owner assignment for comparison with owner assignments that have already been made manually in Coverity Connect to evaluate the accuracy of the owner assignment rules. In this case, you must use the `--no-trage-filters` option.

--owner-assignment-rules <RULE1>,<RULE2>,...,<RULEn>
: Determines an owner by consulting history from the SCM system. If this option is specified, you must include at least one <rule>. Multiple <rule> options must be comma-separated. If this option is not specified, then all rules are applied. In the descriptions below, main event refers the event that Coverity Connect first highlights when the user clicks on the defect in the defects list.

  The rules are:

  - `file` - Queries the SCM for the person that most recently modified the file containing the main event, and that SCM user is the chosen owner.
  - `line` - Queries the SCM for the person that most recently modified the particular line of code that has the main event.
  - `function` - If there is a function associated with the main event, then `cov-blame` queries the SCM for the person who most recently modified that function. Otherwise, this rule acts the same as the file rule.
  - `top_events` - Retrieves all of the lines of code that contain a non-interprocedural defect event in the issue, then returns the person that most recently modified any of those lines.
  - `all_events` - Similar to the `top_events` rule, except that it also considers all interprocedural defect events.

- `all_functions` - Combines the `function` and `all_events` rules to query for the person who most recently modified the functions associated with all the defect events. If there are no functions at all, this rule behaves like the `all_files` rule.
- `all_files` - Combines the `file` rule with the `all_events` rule to query for the person who most recently modified the files containing all of the defect events.
- `default_component_owner` - Reports the issue's owner as the designated default owner for the component in Coverity Connect. The output of this rule, unlike all of the other rules, is a Coverity Connect user, and not an SCM user. This rule can yield no assignment if a component does not have a default owner.

--preview-report <`filename`>
Specifies the path and name of the preview report generated by `cov-commit-defects --preview-report-v2`. This option is required.

--scm <`scm_type`>
Specifies the name of the source control management system:

- For Accurev, the property is: `accurev`
- For ClearCase, the property is: `clearcase`
- For CVS, the property is: `cvs`
- For GIT, the property is: `git`
- For Mercurial, the property is: `hg`
- For Perforce, the properties are: `perforce|perforce2009`

  Use `perforce` for versions later than 2010.1. Use `perforce2009` for all perforce versions from 2010.1 and earlier.
- For SVN, the property is: `svn`
- For Team Foundation Server, the property is: `tfs{2008|2010|2012|2013|2015}`

  For example, for version TFS 2013, use `tfs2013`. This option must match the version of TFS that you are using. These TFS options are only supported on Windows.

For usage information for the `--scm` option, see `cov-extract-scm` .

--scm-command-arg <`scm_command_arg`>
Specifies additional arguments that are passed to the command that retrieves the last modified dates. This option can be specified multiple times.

For usage information for the `--scm` option, see `cov-extract-scm` .

--scm-project-root <`scm_root_path`>
Specifies a path that represents the root of the source control repository. This option is only used when specifying `accurev` as the value to `--scm`. When this is used, all file paths that are used to gather information are interpreted as relative to this project-root path.

For usage information for the `--scm` option, see `cov-extract-scm` .

--scm-tool <scm_tool_path>
Specifies the path to an executable that interacts with the source control repository. If the executable name is given, it is assumed that it can be found in the path environment variable. If it is not provided, the command uses the default tool for the specified `--scm` system.

For usage information for the `--scm` option, see `cov-extract-scm` .

--scm-tool-arg <scm_tool_arg>
Specifies additional arguments that are passed to the SCM tool, specified in the `--scm-tool` option, that gathers the last modified dates. The arguments are placed before the command and after the tool. This option can be specified multiple times.

For usage information for the `--scm` option, see `cov-extract-scm` .

--stop-after <limit>
Stops computing owner assignments after a certain number of defects, as specified in <limit>. This allows you to quickly experiment with rules without waiting for a long time for each defect to be assigned an owner.

Shared options

--debug, -g
Turn on basic debugging output.

--ident
Display the version of Coverity Analysis and build number.

--info
Display certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

--tmpdir <tmp>, -t <tmp>
Specify the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>
Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.

- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

See Also

cov-commit-defects

cov-import-scm

cov-extract-scm

Name

cov-build, , cov-build-sbox Intercept all calls to the compiler invoked by the build system and capture source code from the file system.

Synopsis

C, C++, Java, C#, JavaScript, PHP, Python:

`cov-build` --dir <intermediate_directory> | --emit-server <emit_servername:port> | --da-broker <broker_servername:port> [--capture-ignore <program.extension>] [--c-coverage | --cs-coverage | --java-coverage] [--fs-capture-list <file>] [--fs-capture-search <directory>] [OPTIONS] BUILD_COMMAND | --no-command

`cov-build-sbox` --dir <intermediate_directory> | --emit-server <emit_servername:port> | --da-broker <broker_servername:port> [--capture-ignore <program.extension>] [OPTIONS] <BUILD_COMMAND>

Description

The `cov-build` or `cov-build-sbox` command intercepts all calls to the compiler invoked by the build system and captures source code from the file system. Note that parallel builds for ASP.NET applications cannot be virtualized directly with `cov-build`.

Note

Only use the `cov-build-sbox` command if you are using the Scratchbox compiler.

To run Coverity Analysis using the Scratchbox compiler, install Coverity Analysis in the `/scratchbox/users/$USER/host_usr` directory and use the `cov-build-sbox` and `cov-configure-sbox` commands. Exit Scratchbox to perform all other operations.

Note

On MacOS X 10.11, System Integrity Protection (SIP) must be disabled for `cov-build` to work.

Usually the `cov-build` or `cov-build-sbox` command name and option can prefix the original build command. However, if the build command depends on features of the command shell that usually invokes it, such as certain shell variables or non-alphanumeric arguments, you can invoke it using a wrapper script. This preserves the original behavior because the build command is again invoked directly by the kind of shell on which it depends.

For example, if the normal invocation of a Windows build is:

```
> build.bat Release"C:\Release Build Path\"
```

use `cov-build` as follows:

```
> cov-build --dir <intermediate_directory> <wrapper.bat>
```

where <wrapper.bat> is an executable command script that contains the original and unmodified build command.

On Windows, specify both the filename and extension for the build command when using `cov-build`. For example:

```
> cov-build --dir <intermediate_directory> custombuild.cmd
```

Because `cov-build` uses the native Windows API to launch the build command, the appropriate interpreter must be specified with any script that is not directly executable by the operating system. For example, if the normal invocation of a build within Msys or Cygwin is:

```
> build.sh
```

prefix `cov-build` with the name of the shell:

```
> cov-build --dir <intermediate_directory> sh build.sh
```

Similarly, if a Windows command file does not have Read and Execute permissions, or if you want `cov-build` to report the command file's `%ERRORLEVEL%`, explicitly invoke it as a `cmd.exe` command string. For example, to run the Java builder `ant.bat`:

```
> cov-build --dir <intermediate_directory> cmd /c "ant && exit/b"
```

Note

If you run `cov-build` more than once, only the last build's metrics are saved.

Note

If you change the set of compilation options used by your build process, delete the <intermediate_directory> and capture a full build from scratch. Otherwise, the translation units captured using the old options will remain in the emit. The new translation units will not replace the old translation units due to the changed compilation options.

C, C++, and/or C# build capture

For C, C++, and C# source code, after the compiler calls have been intercepted, `cov-build` captures the compiler command line and other information and invokes cov-translate to translate the native command line into one appropriate for `cov-emit` or `cov-emit-cs`, which in turn parses and emits the code. Before running `cov-build`, you need to configure your compiler (such as gcc or msvc) by using the `cov-configure` command.

For C# only, if an ASP.NET Web application is detected, `cov-build` will attempt to run `Aspnet_compiler.exe` on the Web application. The output of `Aspnet_compiler.exe` is required by the C# security checkers.

The `cov-build` command creates a log file called `build-log.txt` in the intermediate directory. This log file shows each command that is intercepted, including compiler invocations. For each compiler invocation, the call to `cov-translate` and `cov-emit` are shown, along with any parsing errors and any other compilation errors.

The `cov-build` command intercepts compiler invocations for single-threaded builds and parallel builds on a single machine. Distributed builds, which use remote procedure calls or some other protocol to

invoke builds or compilations on several machines, cannot be virtualized directly with `cov-build`. Contact Coverity support for assistance.

On UNIX, `cov-translate` is invoked before each native compiler invocation. On Windows, `cov-translate` is invoked after each native compiler invocation.

C# build capture is only supported on Windows.

The `cov-build` command expects the configured C and C++ compilers to be those used by the build. The analysis might be skewed if different compiler versions are used. So, consider values that the build scripts and makefiles might define for `$PATH`, `$CC`, and so on. If compiler pathnames are unknown, configure a template.

See the section called “Build and filesystem capture examples”.

Java build capture

The `cov-build` command works somewhat differently for Java than for C, C++, and C#. In addition to gathering and compiling source files, `cov-build` for Java also collects compiled files and any JAR files or class files in the classpath.

For Java, the command also runs the compiler in debug mode so that Coverity Analysis can analyze the compiled code. This automatic behavior is equivalent to running `javac -g` and also the same as setting `debug="true"` in an Ant compile task.

Note

You must use supported compilers when using `cov-build` with Java. For details, see the *Coverity 8.0 Installation and Deployment Guide*.

You can use the `--config` option to `cov-configure` and `cov-build` to establish and maintain separate configuration directories for each language.

The `cov-build` command expects the build to use the configured Java compilers. Compile commands with different pathnames will not be analyzed because `cov-build` cannot identify the compiler version. So, consider which compilers the build scripts and tools might invoke. For example, `ant` can refer to `$JAVA_HOME` or a default pathname (not `$PATH`) to find the `java` command. When using `ant` on the Mac OS, you need to set the `$JAVA_HOME`. Otherwise, the Mac OS will select something other than what `cov-configure` will set. If compiler pathnames are unknown, configure a template.

Note

When running 64-bit Coverity Analysis tools against a 32-bit Java SDK, `cov-build` may fail to capture compilations. Use `--instrument` to work around the issue.

See the section called “Build and filesystem capture examples”.

Filesystem capture for interpreted languages

The command uses `--fs-capture-search` and/or `--fs-capture-list` to generate a list of files for JavaScript, Python, or PHP files that the command captures into the intermediate directory.

See the section called “Build and filesystem capture examples”.

Build and filesystem capture examples

- For compiled languages (build capture):

```
> cov-build --dir <intermediate_directory> <BUILD_COMMAND>
```

- For scripts and interpreted code (filesystem capture):

```
> cov-build --dir <intermediate_directory> --no-command \
  --fs-capture-search <path/to/source/code>
```

  If you are only performing a filesystem capture and not also performing a build capture, you need to pass the `--no-command` command. For more details, see *Coverity Analysis 8.0 User and Administrator Guide*.

- For a combined source code capture (build and filesystem capture):

```
> cov-build --dir <intermediate_directory> \
  --fs-capture-search <path/to/source/code> <BUILD_COMMAND>
```

  Note that the build command must be specified last.

Options

--add-arg <arg>
: Specifies an `--add-arg <arg>` option to the `cov-translate` invocations that are launched by `cov-build`. See the description of this option in the cov-translate command documentation.

--append-log
: Append a log of the current `cov-build` run to an existing build log file, instead of performing the default behaviour, which is to overwrite the log file. For details about the file, see the section called “C, C++, and/or C# build capture”.

--auto-diff
: Compatible with C and C++ builds only. Have `cov-translate` attempt to diff preprocessed files when a compilation fails. See the description of --auto-diff in cov-help cov-translate.

--build-description <string>
: Provides an optional description of the build. `<string>` represents the description. The description is used in the build ID. Some examples might include using a tag:

```
cov-capture ... --build-description linuxbuild-rev1
```

--build-id <build-id>
: Specifies the build ID to use for the test capture run. You can specify only one of the following options: `--build-id-file`, `--build-id-dir`, or `--build-id`.

  The build ID is a string that uniquely identifies a build. Every intermediate directory is associated with a build ID. The build ID string has the following format:

```
<user.specified.string>-<our.unique.md5>
```

The MD5 portion is an MD5 hash which identifies the build. Users can specify an additional string to prepend to this hash, which helps identify the build in a human-readable fashion. This additional string can be specified during the invocation of `cov-build`. For example:

```
cov-build --dir idir --c-coverage gcov --build-description \
  "linux-build-tag-1234" make build
```

would identify the build with a build ID of the form:

```
linux-build-tag-1234-<md5-of-this-covbuildinvocation>
```

Generally, the user-specified portion of the build ID should contain information specific to the build that the user is interested in. Examples:

- Build platform - If you build the same codebase on different platforms, this tag can be used to identify the platform used for a particular build.
- Revision control tag - If you build different revisions of a codebase, this tag can be used to identify the tag associated with a particular build. A date may also be used here.

If you want to run `cov-build` multiple times on a single intermediate directory (for example if multiple commands need to be run to generate a build) and you do not want a new build ID generated for each invocation, use --no-generate-build-id.

The build ID for a given intermediate directory can be determined using `cov-manage-emit`, for example:

```
cov-manage-emit --dir idir query-build-id
```

--build-id-dir <dir>
: Identifies a directory that contains `build-id.txt`, a file that specifies a build ID [p. 46]. The `--build-id-dir` option is exactly like `--build-id-file <dir>/build-id.txt`. It is useful when the intermediate directory with the appropriate build ID is available locally, but the `--dir` option is not used.

  You can specify only one of the following options: `--build-id-file`, `--build-id-dir`, or `--build-id`.

--build-id-file <filename>
: Specifies a filename (including the file path to the file) that contains the build ID [p. 46] to use for this run. You can specify only one of the following options: `--build-id-file`, `--build-id-dir`, or `--build-id`.

--build-id-output <filename>
: Writes the build ID that is generated for this run to the specified filename. This provides an easy way to transfer the build ID to different machines when remotely collecting coverage. For example:

```
cov-build --dir idir --c-coverage gcov --build-description \
```

```
  "mybuild" --build-id-output write-build-id-here.txt make
```

--bullseye-dir <dir>
Compatible with C and C++ builds only. Specifies the location of the Bullseye tools for use with `cov-build` or `cov-capture`. This option should point to the location of BullseyeCoverage installation directory. For example, `/opt/bullseye`.

This option must not include the `/bin` portion of the directory.

--bullseye-lib-dir <dir>
Used in conjunction with `--c-coverage bullseye-small` to specify the directory where the small runtime was built. This has no effect with `--c-coverage bullseye`, and will produce a warning.

--c-coverage <tool>
Compatible with C and C++ builds only. Enables C and C++ coverage collection using the specified coverage tool. <tool> represents the coverage tool. The options are `gcov`, `bullseye`, `bullseye-small`, and `function`.

Using the `bullseye` or `bullseye-small` options require a valid BullseyeCoverage installation.

- The following example demonstrates command line usage for `gcov`:

```
cov-build --dir idir --c-coverage gcov make build
```

- The following example demonstrates command line usage with Bullseye. With this option, you must also specify the `--bullseye-dir` option:

```
cov-build --dir t1 --c-coverage bullseye --bullseye-dir /opt/bullseye make build
```

- The following example demonstrates using a Bullseye small runtime with Test Advisor with `bullseye-small`. With this option, you must also specify the `--bullseye-dir` and `--bullseye-lib-dir` options:

```
cov-build --dir idir --c-coverage bullseye-small \
    --bullseye-dir /path/to/bullseye  \
    --bullseye-lib-dir <output-dir>/lib make build
```

Note

You must build the Bullseye small runtime before you can use it with `cov-build` or `cov-capture`. On Windows, the Bullseye small runtime only supports MSVC-based builds (for example, Visual Studio cl).or more information, see *Coverity Test Advisor and Test Prioritization 8.0 User and Administrator Guide*.

You can use `c-coverage` along with `java-coverage` and/or `cs-coverage`, for example:

```
cov-build --dir idir --c-coverage gcov --java-coverage cobertura make build
```

- The following example demonstrates command line usage for Coverity Function Coverage Instrumentation:

```
cov-build --dir t1 --c-coverage function make build
```

--capture
Run the specified build command, and capture the actions and translation units in the `<intermediate_directory>` that is created by the `--initialize` option.

Alternatively, a build system can directly invoke `cov-translate`, possibly in several concurrent processes.

For C and C++ analysis, you can run concurrent, distributed builds across multiple machines with the `--capture` option if the `<intermediate_directory>` is located on an NFS partition. Distributed builds are only supported on Linux and Solaris systems.

--capture-ignore <program-name-with-extension>
On Windows 2000 only, use this option to specify the base name (including the extension) of a program invoked by the native build that does not exit during the build, such as a service or a daemon. Otherwise, `cov-build` will hang at the end the native build waiting for these programs to terminate. `cov-build` is already aware of NTVDM.EXE and MSPDBSVR.EXE. The program name is case insensitive.

--chase-symlinks
Compatible with C and C++ builds only. Follow symbolic links when determining filenames to report.

--chcmdline-type <type>
Do not use this option unless directed by Coverity Support. This option will be deprecated and removed in a future release.

--coverage-instrumentation-error-threshold <percentage>
The `--coverage-instrumentation-error-threshold` option sets a lower bound for measuring success in instrumenting source code for function coverage. The default threshold is 95 percent. If the instrumentation success rate falls below this threshold, instrumentation is considered unsuccessful, is reported as an error, and `cov-build` returns a non-zero exit code. This option can only be used in combination with the `--c-coverage function` option.

The following example demonstrates the `--coverage-instrumentation-error-threshold` option with a threshold of 60%:

```
cov-build --dir t1 --c-coverage function \
  --coverage-instrumentation-error-threshold 60 make build
```

--cs-coverage <tool>
Compatible with C# builds only. Enables C# coverage collection using the specified coverage tool, represented by `<tool>`. Currently, the only coverage tool option is `opencover`.

Running this command option will cause Test Advisor to attempt to register the profiler DLL in order to collect coverage. In some situations, this might fail. To work around this, you will need to manually register the profiler DLL and then use the `--cs-no-register-profiler` on the command line to prevent the command from trying to register the profiler DLLs again. See `--cs-no-register-profiler` for instructions.

If you are running from a Cygwin shell you might see an error similar to the following:

```
Unhandled Exception: System.ArgumentException: Item has already been added.
```

```
Key in dictionary: 'TMP'  Key being added: 'tmp'
```

To avoid this, `unset` the Cygwin versions of the environment variables, for example:

```
% unset tmp
% unset temp
```

You can use `cs-coverage` along with `java-coverage` and/or `c-coverage`.

--cs-filter <filters>
Used in conjunction with `--cs-coverage opencover` to specify a list of filters to OpenCover. Specifying a filter allows you to control what assemblies are instrumented for coverage collection. For example:

```
+[*]* -[Test*]*
```

This filter would instrument everything except for those modules that start with 'Test'. For more information on filters, see the OpenCover documentation.

--cs-no-register-profiler
Instructs Test Advisor to NOT attempt to automatically register the profiler DLL. If you use this argument, you must manually register the profiler DLLs. To do so, use `regsvr32`:

```
        % regsvr32 <install_dir>/bin/x86/OpenCover.Profiler.dll
        % regsvr32 <install_dir>/bin/x64/OpenCover.Profiler.dll
```

This option should be used when you want to run multiple `cov-build`/`cov-capture` with the `--cs-coverage opencover` option.

You can change the `regsvr32` command to only register per-user, for example:

```
% regsvr32 /n /i:user <install_dir>/bin/x86/OpenCover.Profiler.dll
        % regsvr32 /n /i:user <install_dir>/bin/x64/OpenCover.Profiler.dll
```

You only need to register the x64 DLL if you are on a 64-bit platform.

--cs-test-config <properties-xml-file>
Used in conjunction with `--cs-coverage` to provide the location of a user-created C# test properties file (`<properties-xml-file>`). This file configures per-test separation for C# coverage. If both `--cs-test` and `--cs-test-config` are specified, the value given to `--cs-test-config` takes precedence.

--cs-test <test-framework-name>
Used in conjunction with `--cs-coverage` to select a pre-configured C# test properties file (`<test-framework-name>`). This file configures per-test separation for C# coverage. Valid values are:

- `nunit` - for NUnit tests.
- `xunit` - for XUnit tests.
- `mstest` - for MSTest tests.

Test Advisor provides three configuration files that specify test boundary configuration for the popular test execution frameworks: MSTest, NUnit and XUnit. The properties files are located in the following directory (but note that in the option parameter, you only need to specify the name, not the entire path):

```
<install_dir_ca>/config/
```

--cygpath <path>
Specify the path to the directory, which contains the bin directory of the Cygwin installation, if it is not in the PATH environment variable.

--cygwin
Compatible with C and C++ builds only. On Windows, indicates that the build is done with Cygwin. This option allows Cygwin-style paths to be used in the native build command. However, you must use Windows-style paths for all Coverity Analysis commands.

--da-broker <broker_servername:port>
[Coverity Dynamic Analysis option] Specifies the Coverity Dynamic Analysis broker server to receive Coverity Dynamic Analysis results for this run. You can specify a hostname or an IP address. The port is optional (the default is `4422`). Currently, only IPv4 addresses are supported. This option is only used in conjunction with `--java-da` and cannot be combined with `--dir`, but using `--log-dir` is recommended for diagnostic purposes.

See also, `--java-da-opts`.

--delete-stale-tus
Automatically deletes translation units that are created from source files that were renamed or removed. This capability is off by default. Use this command when you perform an incremental build when you have deleted/renamed source files.

--desktop
The `--desktop` option can be used when running `cov-build` in preparation for Desktop Analysis. It behaves similarly to --record-only for C/C++ builds, and disables bytecode decompilation in Java and C# builds. `--desktop` can be paired with `--record-with-source` in order to capture header files.

Please note that this option is supported for backward compatibility. The preferred method for capturing a build for Desktop Analysis is the `cov-run-desktop --build` option.

--dir <intermediate_directory>
Pathname to an intermediate directory that is used to store the results of the build and analysis.

Exactly one of the options `--emit-server`, `--da-broker`, or `--dir` must be specified. Coverity recommends that you use `--log-dir` when you are not using `--dir`.

--disable-aspnetcompiler
[C# builds only] Disables the automatic invocation of `Aspnet_compiler.exe` for any ASP.NET Web applications that are detected in the build. The output of `Aspnet_compiler.exe` is required by the C# security checkers.

Use this option if you are manually running `Aspnet_compiler.exe` as part of your native build or as part of your Coverity Analysis workflow. For further information, see "Capturing an ASP.NET Web application" in Coverity Analysis 8.0 User and Administrator Guide.

--disable-compute-coverability
Skips computing coverability when you run a build or capture. This option is useful if the test will be run in a separate step. Using this option can speed up the build process.

--disable-cs-parse-error-recovery
Disables extra attempts to recover from problems parsing C# source. Though this recovery process takes some extra time, it greatly reduces the impact of parsing problems in most cases. It works by attempting to emit subsets of the input files that would not otherwise reach the emit database.

Typically, this option is needed only if the recovery process takes too long and becomes unmanageable. Note that unlike for Java, this mode is enabled by default for C# because it uses a more efficient algorithm.

--disable-gcov-arg-injection
This option is compatible with C and C++ builds only, and is used in conjunction with `--c-coverage gcov`. It prevents `cov-build` command from automatically injecting the `gcov` command line arguments into the native build. This is useful if you want to use your build system's own `gcov` targets instead of having this command run it automatically. For example:

```
cov-build --dir idir --c-coverage gcov --disable-gcov-arg-injection make gcov
```

--disable-java-per-class-error-recovery
Disables per-class error recovery, which is the default behavior of `cov-build`. Enabling per-class error recovery increases the percentage of source files that can be emitted by attempting to use the class files of the source files that cause parse errors. This default behavior could potentially increase the time to emit files, but it is usually faster than `--enable-java-per-file-error-recovery` because it does not try to emit each file one at a time.

Per-class error recovery is unlikely to correct `cov-emit-java` crashes. It also requires the presence of output class files. To address such issues, see `--enable-java-per-file-error-recovery`.

--disable-local-emit-server
Prevents the command from starting the emit server (as part of the automated startup process). This option is not for general use and should only be used if startup problems occur. This option only takes effect when `--emit-server` is specified.

--emit-cmd-line-id
This option is deprecated as of the 4.4 release.

--emit-parse-errors
Deprecated. Compatible with C and C++ builds only. Uses the `--enable PARSE_ERROR` option in `cov-analyze` instead. Specifies that compile failures should be made visible in Coverity Connect, appearing as defects.

--emit-server <emit_servername:port>
Specifies the emit server to send coverage to for this run. You can specify a hostname or an IP address. The port is optional (the default is 15772). Currently, only IPv4 addresses are supported.

This option is only used in conjunction with `--c-coverage gcov`.

--enable-java-parse-error-recovery
Enables the error recovery algorithm that produces the highest emit percentage in most cases. Currently, this option enables per-class error recovery, but its behavior might change in future. Note that this algorithm is also enabled by default if no error recovery option is specified on the command line.

Explicitly enabling an error recovery algorithm on the command line will automatically disable any incompatible error recovery algorithms.

--enable-java-per-class-error-recovery
[Deprecated as of 7.6.0] The `cov-build` command now performs per-class error recovery by default, so it is no longer necessary to use this option.

For information about per-class error recovery and about how to disable it, see `--disable-java-per-class-error-recovery`. See also, `--enable-java-per-file-error-recovery`.

--enable-java-per-file-error-recovery
Use this option if your native Java compiler is able to compile your source code successfully, but `cov-emit-java` crashes. This option might also help when `cov-emit-java` has parse errors and there are no class files available for per-class error recovery.

--enable-pch
Compatible with C and C++ builds only. Tells `cov-emit` to use pre-compiled header files when the Microsoft Visual Studio compiler uses them. This is not strictly necessary for a Microsoft Visual Studio project that uses pre-compiled headers, but should improve performance. This option does not work with the `--instrument` option and is only supported on 64-bit platforms.

--encoding <enc>
Compatible with C and C++ builds only. Specifies the encoding of source files. Use this option when the source code contains non-ASCII characters so that Coverity Connect can display the code correctly. The default value is US-ASCII. Valid values are the ICU-supported encoding names:

US-ASCII

UTF-8

UTF-16

UTF-16BE
UTF-16 Big-Endian

UTF-16LE
UTF-16 Little-Endian

UTF-32

UTF-32BE
UTF-32 Big-Endian

UTF-32LE
: UTF-32 Little-Endian

ISO-8859-1
: Western European (Latin-1)

ISO-8859-2
: Central European

ISO-8859-3
: Maltese, Esperanto

ISO-8859-4
: North European

ISO-8859-5
: Cyrillic

ISO-8859-6
: Arabic

ISO-8859-7
: Greek

ISO-8859-8
: Hebrew

ISO-8859-9
: Turkish

ISO-8859-10
: Nordic

ISO-8859-13
: Baltic Rim

ISO-8859-15
: Latin-9

Shift_JIS
: Japanese

EUC-JP
: Japanese

ISO-2022-JP
: Japanese

GB2312
: Chinese (EUC-CN)

ISO-2022-CN
: Simplified Chinese

Big5
: Traditional Chinese

EUC-TW
: Taiwanese

EUC-KR
: Korean

ISO-2022-KR
: Korean

KOI8-R
: Russian

windows-1251
: Windows Cyrillic

windows-1252
: Windows Latin-1

windows-1256
: Windows Arabic

Note

If your code is in SHIFT-JIS or EUC-JP, you must specify the `--output_object_encoding SHIFT-JIS` or `--output_object_encoding EUC-JP` option (respectively) for `cov-emit` in order to avoid receiving STRING_OVERFLOW false positives.

--finalize, -fin
: Compatible with C and C++ builds only. Combines the build log and metrics from all the host machines that ran `cov-build` with the same `<intermediate_directory>`, and indicate any additional steps that are needed to prepare for a C and C++ analysis.

  Do not specify a build command when using the `--finalize` option.

--force
: Specifying this options causes the Coverity compiler to attempt all source files, including files that have already been emitted and whose timestamps have not changed. This is equivalent to `--force` in the respective compiler, for example `cov-emit`.

--fs-capture-just-print-matches
: This option is for debugging purposes only.

  Suppresses the emission of files matched with filesystem capture, and outputs a list of files that would have been emitted. This should generally be used with `--no-command`.

--fs-capture-list <file>
Specifies a file that contains a list of files to use as analysis inputs. Files that are recognized as useful for analysis, such as those matching patterns established in calls to `cov-configure`, will be emitted into the intermediate directory. For example, `cov-configure --python` marks `*.py` as useful for analysis, and `cov-configure --javascript` marks `*.js`, `*.html`, and `*.htm` as useful.

This option is a lower-level supplement to or an alternative to --fs-capture-search.

A recommended way of using this option is to provide a list of revision-controlled files as the potential analysis inputs. Example for a git repository:

```
git ls-files > scm_files.lst
cov-build --fs-capture-list scm_files.lst <other options>
```

Note that this option can be specified more than once to include multiple lists of analysis inputs.

See also, --fs-capture-search and --no-command.

--fs-capture-search <directory>
Recursively searches all files in the specified directory for analysis inputs. Files that are recognized as useful for analysis, such as those matching patterns established in calls to `cov-configure`, will be emitted into the intermediate directory. For example, `cov-configure --python` marks `*.py` as useful for analysis, and `cov-configure --javascript` marks `*.js`, `*.html`, and `*.htm` as useful. However, this option ignores any file matching `--fs-capture-search-exclude-regex` and certain files and directories when the specified directory is in the path:

- Symlinks are ignored unless the specified directory is itself a symlink or is located under a symlink.
- Files and directories starting with a period (`.`), including their contents, are ignored unless the specified directory itself starts with a period or is located under such a directory.
- Directories named SCCS, including their contents, are ignored unless the specified directory itself is named SCCS or is located under such a directory.
- Coverity intermediate directories and their contents are ignored.

Note that this option can be specified more than once to search multiple directories for analysis inputs.

See also, --fs-capture-list, --fs-capture-search-exclude-regex, and --no-command.

--fs-capture-search-exclude-regex <regex>
This option only applies when `--fs-capture-search` is used, otherwise it will result in an error.

While `--fs-capture-search` searches its specified directory for analysis inputs, any files or subdirectories that match the specified regex will be excluded from the search, and thus will not be included in the analysis (unless they are independently selected by the `--fs-capture-list` option). Excluded directories will not have their contents searched for further matches, unless a subdirectory is specifically passed to `--fs-capture-search`.

Note

Regular expressions are searched in full absolute paths, in host format. Thus, matching forward slash without also matching backslash, or vice-versa, will make the pattern platform-specific. For example, to exclude JavaScript files starting with 'mock', use the following (assuming bash-style single quoting):

```
--fs-capture-search-exclude-regex '[/\\]mock.*[.]js$'
```

Without the leading '`[/\\]`', a file called 'hammock.js' would be excluded. Similarly, to exclude all contents of directories named 'test' with parent directory named 'src', use the following:

```
--fs-capture-search-exclude-regex '[/\\]src[/\\]test$'
```

--initialize, -init
Compatible with C and C++ builds only. Creates the specified `<intermediate_directory>` that a set of subsequent builds will use. You can only use this option once, and without a build command, before a parallel build.

--instrument
Compatible with Java, C, C++, and C# builds on Windows only, uses the instrumentation mode instead of the debugger. For certain builds, this configuration can significantly improve build times. In particular, parallel builds will benefit most from `--instrument`.

Known issues and workarounds:

- If Visual Studio (2010 or newer) is running `Tracker.exe`:
  - To avoid losing some intermediate directory files, it is necessary to move or rename your intermediate directory before cleaning your project or solution. The Microsoft tool, `Tracker.exe`, tracks all file reads/writes and then uses this list when cleaning up. As a consequence, intermediate directory files that are on this list will be removed when cleaning your project or solution.
  - The template compiler configuration can cause link failures in the build. To work around this issue, you can take either of the following actions:
    - Generate a non-template compiler configuration.
    - Disable file tracking in your build. If you use `msbuild` to build, you can disable the tracker by adding `/p:TrackFileAccess=false` to your command line. If you use `devenv` to build, you need to add the configuration value to your solution/project files.

  The `cov-build` command issues a warning if it detects `Tracker.exe`.
- Some .NET binaries are 32-bit executables that run as 64-bit. One example is the Visual Studio binary `xdcmake.exe`. Starting with Visual Studio 2015, `csc.exe` is another such binary. If your build uses one of these binaries, your build will fail when using `--instrument`. To work around this issue, you can use the `--treat-as-64bit` argument:

  ```
  cov-build --treat-as-64bit xdcmake.exe --treat-as-64bit csc.exe \
  ```

```
--treat-as-64bit someotherbin.exe --instrument --dir idir msbuild
```

Alternatively, you can specify this list of binaries in an environment variable:

```
COVERITY_INSTRUMENT_64BIT_EXES=xdcmake.exe;csc.exe;someotherbin.exe
```

Both the `--treat-as-64bit` argument and the environment variable have the same effect.

To determine if a binary is affected by this issue, you can run `corflags.exe` on it and check the output. This tool is installed with Visual Studio and the Windows SDK.

Example that uses the known `xdcmake.exe` from Visual Studio:

```
% corflags xdcmake.exe
Microsoft (R) .NET Framework CorFlags Conversion Tool.
Version  4.0.30319.1
Copyright (c) Microsoft Corporation.  All rights reserved.

Version   : v4.0.30319
CLR Header: 2.5
PE        : PE32
CorFlags  : 9
ILONLY    : 1
32BIT     : 0
Signed    : 1
```

If ILONLY = 1 and 32BIT = 0 in the above output, the affected binary will run as 64-bit on a 64-bit machine. You need add such binaries to COVERITY_INSTRUMENT_64BIT_EXES.

- The capture DLL will still be loaded for persistent processes, even after `cov-build` exits. One such example of a process like this is `mspdbsvr.exe`, which is a special case that is automatically ignored. However, if you find another binary that persists, you can ignore it by adding `--capture-ignore foo.exe` to the `cov-build` command line. It is important to note, however, that you can only ignore the process if it does not start any compilations.
- The `--instrument` argument is incompatible with the __COMPAT_LAYER environment variable. If your environment sets this variable, you must unset it to use `--instrument`.

--java-coverage <tool>
Enables Java coverage collection for Test Advisor using the specified tool. <tool> represents the coverage tool; either `cobertura` or `jacoco`.

You can use `c-coverage` along with `java-coverage` and/or `cs-coverage`, for example:

```
cov-build --dir idir --c-coverage gcov --java-coverage cobertura make build
```

--java-da
[Coverity Dynamic Analysis option] Enables the injection of a Coverity Dynamic Analysis agent into a command used for exercising your program or running tests on it. The `--java-da` option is only compatible with builds of Java code. This option disables build capture and coverage capture, so is incompatible with options related to those tasks. It currently requires the `--da-broker` option to

reference a running Coverity Dynamic Analysis broker and a build id to be specified with one of `--build-id`, `--build-id-dir`, or `--build-id-file`.

Recommendation

Coverity recommends that you use this option with cov-capture instead of `cov-build` because it is not necessary to capture the build.

This option is not currently compatible with `--dir`, but using `--log-dir` is recommended for diagnostic purposes.

See also, the `--java-da-opts` option to this command.

--java-da-opts <da_agent_option_string>
[Coverity Dynamic Analysis option] Adds extra options to the Java agents that are started (see Coverity Dynamic Analysis Agent command-line options and Ant task attributes in the *Coverity Dynamic Analysis 8.0 Administration Tutorial*).

Note that the `cov-capture` command automatically includes options for reaching the Coverity Dynamic Analysis broker, for instrumenting only classes that have been compiled under build capture, and for logging agent diagnostics to files (see `--log-dir`).

See also, the `--java-da` and `--da-broker` options to this command.

Example:

```
--java-da-opts detect-resource-leaks=false,exclude-instrumentation=com.foo.mock
```

--java-instrument-classes <list.csv>
Specifies a file that contains a list of classes to be present on the execution target to execute Java tests on a non-build host. <list.csv> is in CSV file format and has two columns:

- Column 1: The name of the class.
- Column 2: The full path to the source file for the class.

To construct the contents for this file, use the `list-compiled-classes` subcommand to `cov-manage-emit`.

--java-test <config-name>
Used in conjunction with `--java-coverage` to select a pre-configured Java test properties file (`<test-framework-name>`). This file configures per-test separation for Java coverage. Valid values are:

- `junit` - for JUnit3 tests.
- `junit4` - for JUnit tests.

Test Advisor provides two configuration files that specify test boundary configuration for the popular test execution frameworks: `JUnit3` and `JUnit4`. The properties files are located in the following

directory (but note that in the option parameter, you only need to specify the name, not the entire path):

```
<install_dir_ca>>/config/
```

--java-test-config <path-to-config-file>
Used in conjunction with `--java-coverage` to provide the location of a user-created Java test properties file for the Java capture agent. This file configures per-test separation for Java coverage. If both `--java-test` and `--java-test-config` are specified, the value given to `--java-test-config` takes precedence.

--leave-raw-coverage
Used in conjunction with `--c-coverage gcov` or `--java-coverage cobertura.` When this option is specified, this build command will not automatically add the collected coverage data to the emit, but will instead leave it inside the intermediate directory to be used later.

--log-dir <dir>
When `--dir` is not used, this option specifies a directory, such as an intermediate directory, in which to store the build log or capture log file. By default, when neither `--dir` nor `--log-dir` is used, these logs go to a temporary directory that is erased.

--log-server
Compatible with C and C++ builds on Windows only, this argument allows `cov-build` to produce a consistent build log when using `--parallel-emit`. All output in the build log can be attributed to specific executions of each Coverity program. This is the default.

This argument has no effect to and cannot be used in combination with `--instrument`. You will receive an error message.

--merge-raw-coverage <raw-coverage-directory>
Specifies a directory containing raw coverage data that should be added to the emit. When this is specified, a build command is not allowed.

<raw-coverage-directory> is the path to a coverage directory that was previously created with `--leave-raw-coverage.`

You can merge raw coverage from multiple directories in a single invocation by specifying each directory to a separate `--merge-raw-coverage` argument. Each directory to merge may contain any combination of coverage tools that are supported by the OS on which `cov-capture --merge-raw-coverage` is being run.

The following scenario shows the basic usage of `--leave-raw-coverage` `--merge-raw-coverage`:

1. Perform your build.

   ```
   cov-build --dir idir --c-coverage gcov make build
   ```

2. Run the tests, but leave the raw coverage, and copy it to a new location.

   ```
   cov-capture --dir idir --c-coverage gcov --leave-raw-coverage make test
   ```

```
scp -r remote:idir /some/tmp/dir/idir-raw
```

3. Sometime later, merge in the raw coverage.

```
cov-capture --dir idir  --merge-raw-coverage  /some/tmp/dir/idir-raw
```

--merge-raw-coverage-file <tool>:<filename>
Merges raw coverage data in the given file that was generated by the JaCoCO coverage tool. It is also possible to specify test meta properties with the following options:

- Test status with `--teststatus`.
- Test start date/time with `--teststart`.
- Test duration with `--testduration`.
- Suite name with `--suitename`.
- Test name with `--testname`.

--minimal-classpath-emit
Limits the group of emitted JAR files to those needed for compilation of the Java files. The default behavior without this option is to emit all the JAR files in the classpath regardless of whether they are referenced by a Java file in the compilation. This option can improve performance of Java builds with large numbers of unused JAR files on the classpath at the risk of not capturing all the dependencies of the those JAR files. For example if `A.java` references `A.jar`, which has dependencies on `B.jar`, this option will prevent `B.jar` from getting emitted even if `B.jar` is on the classpath.

--msbuild-shutdown-maxnodes <N>
For Visual Studio 2010 and newer (only available on Windows platforms): Specifies the maximum number of nodes that `cov-build` should attempt to shut down. Typically, this value is equal to the number of nodes that your Visual Studio project is configured to use. Use this option only if the default behavior is undesirably slow.

--name <name>
Tags a build with a name. This name can then be used can be used for translation unit pattern matching through the `cov-manage-emit build_name` argument.

--no-caa-info
Compatible with C and C++ builds only. Do not collect the information required for Coverity Architecture Analysis in the intermediate directory.

--no-command
Specify this option if there is no command for build or test coverage capture. For example, in JavaScript-only analyses, you specify this option with --fs-capture-search or --fs-capture-list. However, if you are using both build capture and filesystem capture, a single invocation of `cov-build` is recommended, without this option.

--no-error-recovery
Disables source-level error recovery in the parser. This typically should only be used if error recovery is causing problems and you have been instructed to use this option by Coverity support.

--no-generate-build-id
Prevents a new build ID from being generated for every `cov-build` invocation, in the case where you wish to run `cov-build` multiple times on a single intermediate directory.

--no-log-server
Compatible with C and C++ builds on Windows only, this argument forces `cov-build` to revert to its original behavior without the log server, with respect to the build log. This is only intended for use when issues arise using `--log-server`.

--no-msbuild-shutdown
For Visual Studio 2010 and newer (only available on Windows platforms): Disables shutdown of resident msbuilds that are created by the Microsoft Build Engine, `msbuild`. Use this option only if you know that you will not have any msbuild processes running, or if you kill the resident msbuilds through some other method.

--no-network-coverage
Makes the command use the file system instead of the emit server. This option only takes effect when `--c-coverage gcov` is specified.

--no-parallel-translate
Compatible with C and C++ builds only. Disables `cov-translate` parallelization. This will prevent `cov-translate` from running in parallel regardless of the degree of parallelization requested, either directly to `cov-build`, `cov-translate`, through configuration files, or native command line translation.

This can also be added as a `cov-emit` argument in a configuration file (it is not actually passed to `cov-emit`). For example:

```
<prepend_arg>--no-parallel-translate</prepend_arg>
```

--no-preprocess-next
Compatible with C and C++ builds only. Disables the `--preprocess-next` option.

--no-refilter, -nrf
Compatible with C and C++ builds only. When combined with `--replay`, calls `cov-emit` directly with the previously translated command line arguments, instead of calling `cov-translate` again (which is the default).

This option does not work with MSVC PCH.

--optimize-pch-space
This option can be used to reduce disk space consumption by Coverity precompiled headers (PCH). This is only effective when running `cov-build --replay` or `--replay-from-emit`, with PCH enabled (`--enable-pch`).

Note

Note that this option may slow the build process down, and in some rare cases, may have no effect on disk space consumption.

--parallel-emit
This C, C++, and Windows only argument will allow `cov-emit` processes to run in parallel. For certain builds, this argument can significantly improve build times. This argument is enabled by default (see `--serial-emit`).

This argument has no effect and cannot be used in combination with `--instrument`. You will receive an error message.

--parallel-translate=<number_of_processes>
Compatible with C and C++ builds only. Instructs `cov-translate` to run `cov-emit` in parallel when multiple files are seen on a single native compiler invocation. This is similar to the Microsoft Visual C and C++ `/MP` switch. Specify the `<number_of_processes>` to be greater than zero to explicitly set the number of processes to spawn in parallel, or zero to auto-detect based on the number of CPUs. When specified directly to either `cov-build` or `cov-translate`, this option will override any settings set in configuration files or translated through the native command line.

This can also be added as a `cov-emit` argument in a configuration file (it is not actually passed to `cov-emit`). For example:

```
<prepend_arg>--parallel-translate=4</prepend_arg>
```

--parse-error-threshold <percentage>
The percentage of translation units that must successfully compile for the `cov-build` command to not generate a warning. If less than this percentage compiles, the `cov-build` command will give a warning when the build completes. The default value is 95.

Note

When used in conjunction with `--return-emit-failures`, `cov-build` will return error code 8, in addition to generating the warning, if less than the specified percentage compiled.

--preprocess-first
Compatible with C and C++ builds only. Uses the native compiler to preprocess source files and then invokes `cov-emit` to compile the output of the native processor. By default, `cov-emit` (which is invoked by `cov-build`) otherwise tries to preprocess and parse each source file.

Using this option can address some cases in which hard-to-diagnose causes for macro predefinitions are different, or for header files that cannot be found by `cov-emit`. Usually, `cov-configure` attempts to intelligently guess the native compiler's predefined macros and built-in include directories, but sometimes `cov-configure` guesses incorrectly. Using the `--preprocess-first` option circumvents the problem, but at the cost of losing macro information during analysis. Using `--preprocess-first` does not always work because it requires rewriting the native compiler command line, which the native compiler may or may not like.

See also, `--preprocess-next`.

--preprocess-next
Compatible with C and C++ builds only. Attempts to use `cov-emit` to preprocess source files. If that attempt fails, or if `cov-emit` encounters a parse error, this option preprocesses the files with the native preprocessor, and invokes `cov-emit` to compile the output of the native processor. This

offers the benefit of using the higher-fidelity `cov-emit` preprocessor, while also providing a fallback in case of errors.

This option can be disabled with --no-preprocess-next (the latter has precedence over the former). See `--preprocess-first` for information about the effects of using the native preprocessor.

--record-only, -ro
Compatible with C and C++ builds only. Only record the compiles done during the build, do not attempt to parse and emit the code. Later, `cov-build` can be rerun with `--replay` to actually parse and emit the code.

--record-with-source, -rws
Compatible with C and C++ builds only. Compile translation units far enough to pull in all the `#include` files needed by the compilation, and store these in the emit. Later, `cov-build` can be rerun with `--replay-from-emit` to actually parse and emit the code. See the entry for `--replay-from-emit` for more information and examples.

This argument has no effect to and cannot be used in combination with either `--preprocess-first` or `--preprocess-next`. You will receive an error message.

--replay, -rp
Compatible with C and C++ builds only. Replay the parse and emit steps that were previously recorded for a build in the given intermediate directory. If you specify this option, do not specify a build command. This option can be used to quickly update a previously emitted build if the source files have changed.

--replay-failures, -rpf
Compatible with C and C++ builds only. Only attempt to replay the emit for files that had parsing or other compilation failures.

--replay-from-emit, -rpfe
Compatible with C and C++ builds only. Recompile translation units from source contained within the emit directory. Replaying from the emit will have the same results, regardless of changes to the files in the filesystem (including deletion).

This can be used when translation units were added with normal `cov-build` processes (although this will have no real effect unless `--force` has been passed), or with translation units added with `--record-with-source`.

The advantage of using `--record-with-source` and `--replay-from-emit` is that temporary files (such as created by #import) are captured in the emit, and so projects that use `#import` can be replayed, which they cannot with `--replay`. In addition, it is possible to transport the intermediate directory to a different computer/platform and replay it there.

For example, you can record a build on Windows and transfer the intermediate directory to Linux and replay it there. (You will have to use `cov-manage-emit reset-host-name` to change the host.)

This argument has no effect to and cannot be used in combination with either `--preprocess-first` or `--preprocess-next`. You will receive an error message.

--replay-processes <count>, -j <count>
Compatible with C and C++ builds only. When performing `--replay`, spawn up to `<count>` `cov-emit` processes in parallel (on a single machine).

This option accepts the number of processes, or `auto` which sets the number of replay processes to the number of logical processors in the machine (`-j 0` is also accepted and is the same as `auto`).

--return-emit-failures
The `cov-build` command returns with an error code if an emit failure occurs. The return value is a combination (binary OR) of the following flags:

1: The build returned an error code.

2: The build terminated with an uncaught signal (for example, segmentation fault).

4: No files were emitted.

8: Some files failed to compile. By default, this error code is returned if fewer than 95% of the compilation units compiled successfully. You can change this percentage by using the `--parse-error-threshold` option.

16: Command line error, such as an unrecognized option.

Note

The `cov-build` command always returns an error code if the native build fails.

--serial-emit
This C, C++, and C#, Windows-only argument forces `cov-emit` processes to run in serial. This option is disabled by default (see `--parallel-emit`).

This option has no effect and cannot be used in combination with `--instrument`. Attempts to use the two options together will result in an error message.

--suitename <suitename>
Test suite name to attribute coverage to when using `--merge-raw-coverage-file`.

--system-encoding <enc>
Compatible with C and C++ builds only. Specifies the encoding to use when interpreting command line arguments and file names. If not specified, a default system encoding is determined based on host OS configuration.

See --encoding for a list of accepted encoding names.

This option has no effect when used in conjunction with one of the `--replay` options.

--telemetry-network-error-exit-code <exit code>
Specifies the exit code to use when the process aborts due to it exhausting all retries. This option can be customized to allow for the test harness to appropriately react to these failures.

The default exit code is 19.

For more information, see "Error recovery" in the *Coverity Test Advisor and Test Prioritization 8.0 User and Administrator Guide*.

--telemetry-network-error-max-wait <# of seconds>
Specifies the maximum time to wait when recovering from a network related error. The behavior when encountering an error is to sleep and then retry. It will sleep for 1, 5, 30, 60 seconds, respectively, until the maximum wait time has been reached.

The default maximum wait time is 300 seconds.

For more information, see "Error recovery" in the *Coverity Test Advisor and Test Prioritization 8.0 User and Administrator Guide*.

--telemetry-network-error-log <filename>
When a network error is encountered, by default an error will be written to STDERR which can interfere with some test processes. This argument allows these errors to be redirected to a user-specified file.

For more information, see "Error recovery" in the *Coverity Test Advisor and Test Prioritization 8.0 User and Administrator Guide*.

--test-capture-run-tag <string>
Specifies a custom tag to allow for easy selection of this test capture run. For example:

```
linux-build
```

--test-capture-run-timestamp <timestamp>
Specifies the timestamp to use for the test capture run for this invocation. If it is not specified, the current time will be used, which is the typical use case. This option is only provided as a way for multiple test runs to use the same test capture run.

See Accepted date/time formats for proper formatting of the `<timestamp>` argument.

--test-capture-status-file <file containing status>
Specifies a filename that, at the end of the test run, will contain the status that should be used for this test capture run. The file should contain one of:

- `success`
- `failure`
- `unknown`

Anything other than the above will be interpreted as `unknown`.

--testduration <test duration in milliseconds>
Test duration for the test when using `--merge-raw-coverage-file`.

--testname <testname>
Test name to attribute coverage to when using `--merge-raw-coverage-file`.

--teststart <test_start_date/time>
Test start date/time when using `--merge-raw-coverage-file`. Must be in the format: `yyyy-MM-dd HH:mm:ss`.

--teststatus [pass | fail | unknown]
Test status when using `--merge-raw-coverage-file`.

--treat-as-64bit <exe-name>
Compatible with Java, C, C++, and C# builds on 64-bit Windows with `--instrument` only. Some .NET binaries are 32-bit executables that run as 64-bit. One example is the Visual Studio binary `xdcmake.exe`. Starting with Visual Studio 2015, `csc.exe` is another such binary. If your build uses one of these binaries, you must use `--treat-as-64bit` to inform `cov-build` that the specified executable is one of these. For example:

```
cov-build --dir idir --treat-as-64bit csc.exe --instrument msbuild.exe
```

Failing to provide this information to `cov-build` can cause builds using the `--instrument` flag to fail. Builds that are not run with the `--instrument` flag do not need this information.

Note

Before using this option, be sure to read the known issues and workarounds for the `--instrument` option.

Shared options

--config <`coverity_config.xml`> , -c <`coverity_config.xml`>
Use the specified configuration file instead of the default configuration file located at <`install_dir_sa`>`/config/coverity_config.xml`.

--debug, -g
Turn on basic debugging output.

--debug-flags <flag> [, <flag>, ...]
Controls the amount of debugging output produced during a build. These flags can be combined on the command line using a comma as a delimiter.

Valid flags are `build`, `capture`, `translate`, `translate-phases`. For example, `--debug-flags build, translate`.

--ident
Display the version of Coverity Analysis and build number.

--redirect stdout|stderr,<filename>, -rd stdout|stderr,<filename>
Redirect either stdout or stderr to <filename>..

--tmpdir <tmp>, -t <tmp>
Specify the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>
Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

Exit codes

By default, `cov-build` returns the exit code from the native build. For example, if native build command returns `57`, `cov-build` will return `57`. However, in the case of invalid arguments, `cov-build` can return `16`, which can also occur if the native build returns `16`.

If you pass --return-emit-failures to `cov-build`, the return codes change to match the specified emit failure responses.

See Also

cov-configure

cov-emit

cov-translate

Name

cov-collect-models Gather all C/C++ function models from an analyzed intermediate directory into a single model file.

Synopsis

```
cov-collect-models [--dir <intermediate_directory> | --input] [-of
<file.xmldb>] [--make-dc-config] [--text]
```

Description

The `cov-collect-models` command gathers all of the function models from a C/C++ intermediate directory previously analyzed with `cov-analyze` and collects them into a single output model file. This model file can be subsequently passed to `cov-analyze` with the `--model-file` option.

The primary purpose of `cov-collect-models` is to allow interprocedural information from a full analysis run to be used when analyzing only a small portion of the code base. This usually results in finding some interprocedural errors even when only a small portion of the code base is analyzed, and it also usually helps lower the false positive rate.

Note

To use the derived model file on a Windows SMB network shared drive (for example, when running Coverity Desktop local analyses that use derived models), it is necessary to generate the file on a physical disk, then copy it to the shared drive for read-only access by other processes.

Options

--dir <intermediate_directory>
Pathname to an intermediate directory that is used to store the results of the build and analysis.

--input <file.xmldb>, -if <file.xmldb>
Instead of reading models from an intermediate directory, use this input file. This option can be used more than once. If you specify multiple input files, the models in them are merged together and placed into the output file. Models from input files that are specified first have precedence over those in files that are specified later.

--make-dc-config
[C/C++ only] For a description, see the --make-dc-config option to `cov-make-library`.

--output-file <file.xmldb>, -of <file.xmldb>
The path name for the file to store the collected models.

--output-tag <name>
Use this option if you used it when generating analysis results. See the --output-tag option to `cov-analyze`.

--text
Output the models as text. This format is far less efficient than the standard `.xmldb` format, but is easier to debug.

Shared options

--debug, -g
Turn on basic debugging output.

--ident
Display the version of Coverity Analysis and build number.

--info
Display certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

--tmpdir <tmp>, -t <tmp>
Specify the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

See Also

cov-analyze

cov-make-library

Name

cov-commit-defects Commit Coverity Analysis defect reports to a Coverity Connect database.

Synopsis

`cov-commit-defects`

```
--dataport <port_number> | --port <port_number> | --https-port <port_number>
--dir <intermediate_directory>
--host <host_server_name>
--stream <stream_name>
[--authenticate-ssl]
[--auth-key-file <keyfile>]
[--cva ]
[--description "description"]
[--encryption <requirement_level>]
[--extra-output <path>]
[--output-tag <name>]
[--password <password>]
[--preview-report <filename> | --preview-report-v2 <filename>]
[--scm <scm_type>]
[--scm-tool <scm_tool_path>]
[--scm-project-root <scm_root_path>]
[--scm-tool-arg <scm_tool_arg>]
[--scm-command-arg <scm_command_arg>]
[--snapshot-id-file <filename>]
[--strip-path <path>]
[--target <platform>]
[--ticker-mode <mode>]
[--user <user_name>]
[--version <version>]
[OPTIONS]
```

Description

The `cov-commit-defects` command reads analysis output and source data stored in an intermediate directory and writes them to a Coverity Connect database in a stream that you specify. The data are written as a unit to the database; this unit is a snapshot.

> Note that SCM-related command options (`--scm*`) are used to collect SCM (source code management) data solely for the purposes of automatic ownership assignment (see cov-blame and *Coverity Platform 8.0 User and Administrator Guide*). To display SCM data in the Coverity Connect source browser, use `cov-import-scm` prior to running `cov-analyze`.
>
> This command requires that source files remain in their usual locations in the checked-out source tree. If the files are copied to a new location after checkout, the SCM query will not work.

After you perform the commit, you can view the defects in Coverity Connect alongside the source code that generated them. The issues in the intermediate directory are discovered through the cov-analyze command.

Note

It is possible to use the `cov-build` command to capture Java, C/C++, and C# builds to the same intermediate directory. The number of different-language builds in an intermediate directory dictates the number of times that you need to run the `cov-commit-defects` command.

For example, in the case that you used `cov-build` to capture C/C++ and Java builds to the same intermediate directory, you need to run `cov-commit-defects` twice on the analysis results (obtained from running `cov-analyze` on the builds in that directory). You run `cov-commit-defects` once to commit the Java resources to the appropriate Java stream in Coverity Connect, another time to commit the C/C++ resources to their C/C++ streams.

Options

--authenticate-ssl
: This option is a synonym for `--on-new-cert distrust`, but only one of these options may be present.

--auth-key-file <keyfile>
: Specify the location of a previously created authentication key file, used for connecting to the Coverity Connect server.

--certs <`filename`>
: In addition to CA certificates obtained from other trust stores, use the CA certificates in the given `filename`.

--cid-assignment-timeout <timeout-seconds>
: When committing with `--preview-report` or `--preview-report-v2` to an instance of Coverity Connect in a clustered environment, assignments of CIDs to issues can delay the completion of the preview report. This option specifies the number of seconds to wait for this phase of preview report processing to complete. If assigning CIDs takes longer than <timeout-seconds>, Coverity Connect will leave some of the CIDs unassigned. They will have null values in the preview report. The default CID assignment timeout is 60 seconds.

--comparison-snapshot-id <snapshot-id>
: This option is used in conjunction with the `--preview-report-v2` option to specify the snapshot with which the preview report will compare the commit's defect instances. A boolean flag, called `presentInComparisonSnapshot`, is included in the preview report indicating whether each of this commit's defect occurrences is present in the given snapshot. The default value is the most recent snapshot ID in the specified stream.

--cva
: Generate architectural information for a C/C++ snapshot. After using this option, Coverity Connect users can download this snapshot's `.cva` file. This option is not valid when committing a C# or Java snapshot.

For information about downloading the `.cva` file, see the Architecture Analysis appendix in the *Coverity Platform 8.0 User and Administrator Guide*.

If you are analyzing and committing projects with multiple languages, you must analyze and commit C/C++ code with the `--cva` option separately from analyze and commit commands for other languages. For example, if you have a project with C/C++ and Java, you should run the commands in the following groupings:

1. `cov-analyze` (for C/C++ build)
2. `cov-commit-defects` with `--cva` option

1. `cov-analyze --java` (for Java build)
2. `cov-commit-defects`

Note

The preferred method for creating the `.cva` file is using the `cov-export-cva` command.

--dataport <port_number>
Used with the `--host` option to specify the commit port on Coverity Connect. You can use only one of `--port`, `--dataport`, or `--https-port` to specify the port for commits.

The commit port is determined using one of the following methods, listed in order of priority (the first applicable item will be used):

1. The commit port specified with `--dataport`.
2. The HTTP port specified with `--port`. `cov-commit-defects` connects to this port to retrieve the dataport using HTTP if `--ssl` is absent or HTTPS if `--ssl` is present.
3. The HTTPS port specified with `--https-port`. `cov-commit-defects` connects to this port using HTTPS to retrieve the dataport.
4. The commit port, specified with the `cim/commit/port` element from the XML configuration file.
5. The HTTP or HTTPS port specified with the `cim/port` or `cim/https_port` element, respectively, from the XML configuration file.
6. HTTP port 8080 without `--ssl` or 8443 with `--ssl` is used to retrieve the dataport from Coverity Connect.

--description <description>
Specify a description for the committed snapshot.

--dir <intermediate_directory>
Pathname to an intermediate directory that is used to store the results of the build and analysis.

--encryption <requirement_level>
`cov-commit-defects` uses this option to communicate with Coverity Connect to determine if the dataport connection will be encrypted. By default, the value for `--encryption <requirement_level>` is "preferred".

The available values for `<requirement_level>` are:

required
: The commit will proceed only if the server requires or prefers encryption. The connection will be encrypted.

preferred
: The connection will be encrypted if the server requires or prefers encryption. Otherwise, the connection will be unencrypted

none
: The commit will proceed only if the server prefers encryption or has an encryption setting of none (meaning it requires no encryption). The connection will be unencrypted.

--extra-output <path>, -xo <path>
: [Deprecated] This option is deprecated as of version 5.5 and subject to removal or change in a future release.

  Specify additional output directories from parallel analysis snapshots. Use this option for each output directory, in addition to the default `<intermediate_directory>/output` directory, that you want to commit into a single snapshot ID in the Coverity Connect.

--host <server_hostname>
: Specify the server hostname to which to send the results. The server must have a running instance of Coverity Connect.

  If unspecified, the default is the `host` element from the XML configuration file.

  Note

  If you're running `cov-commit-defects` on a Linux OS, or using `--ssl`, you must enter the full host and domain name for the `--host` parameter:

  ```
  --host <server_hostname.domain.com>
  ```

--https-port <port_number>
: See description for `--dataport`.

--misra-only
: [Deprecated in 8.0] Using this option will result in an error.

--noxrefs
: Tells cov-commit-defects to skip the phase of transferring xrefs (cross-reference data) to Coverity Connect. It is useful for debugging and doing commits where the user doesn't mind that, when viewed in the Coverity Connect, their code lacks cross-reference information. A user might prefer that if they were sensitive to the amount of time taken by commit to execute.

--on-new-cert `<trust | distrust>`
: Indicates whether to trust (with trust-first-time) self-signed certificates, presented by the server, that the application has not seen before. Default is `distrust`. For information on the new SSL certificate management functionality, please see *Coverity Platform 8.0 User and Administrator Guide*

--output-tag <name>
Use this option if you used it when generating analysis results. See the --output-tag option to `cov-analyze`.

--password <password>, -pa <password>
Specify the password for either the current user name, or the user specified with the `--user` option. For security reasons, the password transmitted to the Coverity Connect is encrypted. If unspecified, the default is (in order of precedence):

1. The `password` element from the XML configuration file.
2. The environment variable `COVERITY_PASSPHRASE`.
3. The password in the file pointed to by the environment variable `COVERITY_PASSPHRASE_FILE`.

Note

The passphrase can be stored in a file without any other text, such as a newline character.

Warning

On multi-user systems, such as Linux, users can see the full command line of all commands that all users execute. For example, if a user uses the `ps -Awf` command, identifying information such as usernames, process identities, dates and times, and full command lines display.

This attribute supports the commit process.

--port <port_number>
See description for `--dataport`.

--ssl
Specifies that SSL is to be used for both HTTPS port and dataport connections. For the negotiation with the server on whether to use SSL on the dataport, this is the equivalent of `--encryption required`.

--preview-report <filename>
Instead of sending files, cross-references, and other assets to the server, this option sends only the defect occurrences. The server returns a commit preview report, which is written in JSON format, to `<filename>`.

The commit preview report uses the structure defined in the following table. Note that the order of items contained within objects or arrays is arbitrary.

There is a second version of the preview report, used by Coverity Desktop Analysis, which contains additional details for each CID. See --preview-report-v2 <filename> for more information.

Table 1. Report v1 element syntax

Report element	Comments
report <- { "header" : *header*,	

Report element	Comments

`"analysisInfo" : analysisInfo,`
`"issueInfo" : issueInfo,`
`}`	

`header  <- {`
`"format" : "commit preview report",`
`"version" : 1,`
`}`	

`analysisInfo <- {`
`"command" : string,`
`"reportTimestamp" : string,`
`"user" : string,`
`}`
	`ReportTimestamp` has format `yyyy-mm-ddThh:mm:ss.mmmZ`. The `T` separates the date from the time. The `Z` indicates that the timestamp is in UTC.

`issueInfo <- [`
`{`
`"cid" : number or null,`
`"mergeKey" : string,`
`"occurrences" : occurrences,`
`"triage" : triage,`
`}, ...`
`]`
	Each distinct issue has a unique identifier, the `mergeKey`. The CID may sometimes be null in clustered installations.

`occurrences  <- [`
`{`
`"checker" : string,`
`"file" : string,`
`"function" : string,`
`"extra" : string,`
`"subcategory" : string,`
`"mainEvenLineNumber" : integer,`
`"mainEventDescription" : string,`
`}, ...`
`]`
	Each `issueInfo` has one or more occurrences, all with the same `mergeKey`.

`triage <- {`
`"classification" : string,`
`"action" : string,`
`"fixTarget" : string,`
`"severity" : string,`
`"owner" : string,`
`}`
 	
(one additional item for each custom attribute. Value is *`string`* or `null`.)	

--preview-report-v2 <filename>

Similar to `--preview-report`, this option sends only defect occurrences to the server, which then returns a commit preview report, written in JSON format, to `<filename>`. Version two of the preview report contains all of the information present in version one, with several additional fields.

The commit preview report (v2) uses the structure defined in the following table. Note that the order of items contained within objects or arrays is arbitrary.

Table 2. Report v2 element syntax

Report element	Comments

report <- {
"header" : *header*,
"analysisInfo" : *analysisInfo*,
"issueInfo" : *issueInfo*,
}	

header <- {
"format" : "commit preview report",
"version" : 2,
}	

analysisInfo <- {
"command" : *string*,
"reportTimestamp" : *string*,
"user" : *string*,
"comparisonSnapshotId" : *string*,
"ownerAssignmentRule" : *string*
 	
"ownerLdapServerName" : *string*,
 	
}
	`ReportTimestamp` has format `yyyy-mm-ddThh:mm:ss.mmmZ`. The `T` separates the date from the time. The `Z` indicates that the timestamp is in UTC.
	
	The `comparisonSnapshotId` is the snapshot identifier given by the `--comparison-snapshot-id` command line parameter. If not specified by `--comparison-snapshot-id`, this will be the ID of the most recent snapshot.

issueInfo <- [{ "cid" : *number* or null, "mergeKey" : *string*, "occurrences" : *occurrences*, "triage" : *triage*, "customTriage" : *customTriage*, "presentInComparisonSnapshot" : *boolean*, "firstDetectedDateTime" : *string*, "ownerLdapServerName" : *string*, }, ...]	Each distinct issue has a unique identifier, the `mergeKey`. The CID may sometimes be null in clustered installations. The `presentInComparisonSnapshot` flag is true if this issue occurs in the comparison snapshot identified by the `comparisonSnapshotId` (listed in the analysisInfo element).
occurrences <- [{ "checker" : *string*, "file" : *string*, "function" : *string*, "extra" : *string*, "subcategory" : *string*, "mainEvenLineNumber" : *integer*, "mainEventDescription" : *string*, "componentName" : *string*, "componentDefaultOwner" : *string*, "componentDefaultOwner " *string* }, ...	Each `issueInfo` has one or more occurrences, all with the same `mergeKey`.

Report element	Comments
]	

triage <- {
"classification" : *string*,
"action" : *string*,
"fixTarget" : *string*,
"severity" : *string*,
"owner" : *string*,
"legacy" : *string*,
"externalReference" : *string*,
}	

customTriage <- {
"quotedString" : *string*
}
	The custom triage attributes, if any, are listed here.

--product <product_name>
Deprecated. See `--stream`.

--scm <scm_type>
Specifies the name of the source control management system:

- For Accurev, the property is: `accurev`
- For ClearCase, the property is: `clearcase`
- For CVS, the property is: `cvs`
- For GIT, the property is: `git`
- For Mercurial, the property is: `hg`
- For Perforce, the properties are: `perforce|perforce2009`

  Use `perforce` for versions later than 2010.1. Use `perforce2009` for all perforce versions from 2010.1 and earlier.
- For SVN, the property is: `svn`
- For Team Foundation Server, the property is: `tfs{2008|2010|2012|2013|2015}`

  For example, for version TFS 2013, use `tfs2013`. This option must match the version of TFS that you are using. These TFS options are only supported on Windows.

For usage information for the `--scm` option, see `cov-extract-scm` .

--scm-command-arg <scm_command_arg>
Specifies additional arguments that are passed to the command that retrieves the last modified dates. This option can be specified multiple times.

For usage information for the `--scm` option, see `cov-extract-scm` .

--scm-project-root <scm_root_path>
Specifies a path that represents the root of the source control repository. This option is only used when specifying `accurev` as the value to `--scm`. When this is used, all file paths that are used to gather information are interpreted as relative to this project-root path.

For usage information for the `--scm` option, see `cov-extract-scm` .

--snapshot-id-file <filename>
If the commit succeeds, write the snapshot ID for this commit to the specified file, and make this file writable.

--scm-tool <scm_tool_path>
Specifies the path to an executable that interacts with the source control repository. If the executable name is given, it is assumed that it can be found in the path environment variable. If it is not provided, the command uses the default tool for the specified `--scm` system.

For usage information for the `--scm` option, see `cov-extract-scm` .

--scm-tool-arg <scm_tool_arg>
Specifies additional arguments that are passed to the SCM tool, specified in the `--scm-tool` option, that gathers the last modified dates. The arguments are placed before the command and after the tool. This option can be specified multiple times.

For usage information for the `--scm` option, see `cov-extract-scm` .

--security-file <license file>, -sf <license file>
Path to a Coverity Analysis license file. If not specified, this path is given by the `security_file` element in the XML configuration file, or `license.dat` in the same directory as `<install_dir_sa>/bin`.

--stream <stream_name>
Specifies a stream name to which to commit these defects.

If the stream option is not specified, the `stream` element from the XML configuration file is used.

If the stream is associated with a specific language and you attempt to commit results from other languages to that stream, the commit will fail. However, in Coverity Connect, it is possible to associate a stream with multiple languages even if the stream was previously associated with a single programming language.

--strip-path <path>, -s <path>
Strips the prefix of a file name path in error messages and references to your source files. If you specify the `--strip-path` option multiple times, you strip all of the prefixes from the file names, in the order in which you specify the `--strip-path` argument values.

Note that instead of using this option when committing issues to Coverity Connect through `cov-commit-defects`, you can enhance end-to-end performance by using this option with `cov-analyze` when analyzing code, or with `cov-import-results` when importing third party issues.

--target <target_name>
Target platform for this project (for example, `i386`).

--ticker-mode <mode>
Set the mode of the progress bar ticker. The available modes are:

none
No progress bar is displayed.

no-spin
Only the print stars are displayed; the spinning bar is not.

spin
This is the default mode. Stars with a spinning bar at the end are displayed. Each file, function, or defect committed corresponds to steps of spin.

--user <user_name>
Specifies the user name that is shown in Coverity Connect as having committed this snapshot. If unspecified, the default is:

1. The `user` element from the XML configuration file.
2. The environment variable `COV_USER`.
3. The environment variable `USER`.
4. The name of the operating system user invoking the command (where supported).
5. The UID of the operating system user invoking the command (where supported).
6. `admin`.

--version <version>
This snapshot's project version.

Shared options

--config `<coverity_config.xml>` , -c `<coverity_config.xml>`
Use the specified configuration file instead of the default configuration file located at `<install_dir_sa>/config/coverity_config.xml`.

--debug, -g
Turn on basic debugging output.

--ident
Display the version of Coverity Analysis and build number.

--info
Display certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

--tmpdir <tmp>, -t <tmp>
Specify the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>
Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

Exit codes

This command returns the following exit codes:

- 0: Success.
- 3: Non-fatal error.
- Other: Internal error.

Any errors during a commit are recorded in Coverity Connect in the `<install_dir_cc>/logs/cim.log` file. Errors and warnings are also recorded to the `<intermediate-dir>/output/commit-error-log.txt` file.

Examples

Commit data to host coverity_server1 for the `xalan` stream:

```
> cov-commit-defects --host coverity_server1 --dataport 9090 --stream xalan --dir
 xalan_int_dir \
  --user admin --password 1256
```

Commit data to host coverity_server1 for the and `xalan` stream, using an XML configuration file for all settings except the intermediate directory:

```
> cov-commit-defects  --dir xalan_int_dir --config test_cim_commit.xml
```

The `test_cim_commit.xml` XML configuration file contents are shown next:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE coverity SYSTEM "coverity_config.dtd">
<coverity>
    <config>
        <cim>
            <host>coverity_server1</host>
            <client_security>
                <user>admin</user>
                <password>1256</password>
            </client_security>
            <commit>
                <port>9090</port>
                <source-stream>xalanSource</source-stream>
            </commit>
        </cim>
    </config>
</coverity>
```

The `port` element in this example refers to the commit port (equivalent to the `--dataport` option).

Name

cov-configure,, cov-configure-sbox Create a configuration for a native compiler or scripting language, and generate a `coverity_config.xml` file.

Synopsis

Clang:

```
    cov-configure [--config <cov_config_file>] --clang
```

GNU C/C++ compiler (gcc/g++):

```
    cov-configure [--config <cov_config_file>] --gcc
```

JavaScript:

```
    cov-configure [--config <cov_config_file>] --javascript
```

Microsoft C/C++ compiler (cl):

```
    cov-configure [--config <cov_config_file>] --msvc
```

Microsoft C# compiler (csc):

```
    cov-configure [--config <cov_config_file>] --cs
```

Oracle Java compiler (javac):

```
    cov-configure [--config <cov_config_file>] --java
```

PHP:

```
    cov-configure [--config <cov_config_file>] --php
```

Python:

```
    cov-configure [--config <cov_config_file>] --python
```

Other compiler:

```
    cov-configure [--config <cov_config_file>] [--template]
      --compiler <name> --comptype <type> [--version <comp_version>]
      [--cygpath <path>] [--cygwin] [--force]
      [--xml-option=[tag][@<language>]]
```

Other compiler when using Scratchbox:

```
    cov-configure-sbox [--config <cov_config_file>] [--template]
      --compiler <name> --comptype <type> [--version <comp_version>]
      [--cygpath <path>] [--cygwin] [--force]
      [--xml-option=[tag][@<language>]]
```

Description

The `cov-configure` or `cov-configure-sbox` command creates a configuration for a compiler (or compiler family) and/or a scripting language, such as JavaScript.

Note

Only use the `cov-configure-sbox` command if you are using the Scratchbox compiler.

To run Coverity Analysis using the Scratchbox compiler, install Coverity Analysis in the `/scratchbox/users/$USER/host_usr` directory and use the `cov-build-sbox` and `cov-configure-sbox` commands. Exit Scratchbox to perform all other operations.

By default, if no other configuration file or directory is specified, the configuration is created at `<install_dir>/config/coverity_config.xml`. Each invocation of the `cov-configure` or `cov-configure-sbox` command adds a compiler configuration in its own subdirectory of the directory containing the `coverity_config.xml` file, and `coverity_config.xml` contains an include directive for that compiler-specific configuration.

Note

On some Windows platforms, you might need to use Windows administrative privileges when you run `cov-configure`.

Typically, you can set the administrative permission through an option in the right-click menu of the executable for the command interpreter (for example, `Cmd.exe` or Cygwin) or Windows Explorer.

C/C++ configuration

In general, for C/C++ `cov-configure` tries to make an intelligent guess as to the native compiler's built-in macro definitions and system include directories. For some compilers, such as gcc, there are command line arguments that reveal this information, and `cov-configure` invokes the native compiler to discover this information. For some compilers, there is no standard way of getting this information, so `cov-configure` tries several methods to gather this information. However, these methods are not perfect and sometimes a configuration is generated that is incomplete or incorrect, with the result that some obscure parsing error occurs during the parsing of some source file or header file. Some manual configuration could be necessary. See the compiler information in the *Coverity Analysis 8.0 User and Administrator Guide*.

C# configuration

Use the template configuration, `cov-configure --cs`, unless you have a clear understanding of the alternative C# configuration option.

If you do not use the template configuration, you must specify the location of the C# compiler, `csc`, to the `--compiler` option. If you do not use the template configuration, the `cov-build` command will capture only those C# compile commands that have the same absolute pathname as a compiler that was identified by a preceding `cov-configure` command. Any symbolic links in the pathnames of the configured and executed compile commands will be replaced by real directory names before comparison. For example, in a Unix system, the `inode` of the compile commands must match. Because

the pathnames to the C# compile command can vary, you need to configure all of the pathnames in use or to define a template configuration to handle all of them.

Clang

Use the configuration, `cov-configure --clang`.

Java configuration

Use the template configuration, `cov-configure --java`, unless you have a clear understanding of the alternative Java configuration option.

If you do not use the template configuration, you must specify the location of the Java compiler, `javac`, to the `--compiler` option. This specification configures both the compiler and the virtual machine (`java`, `java.exe`, or `javaw.exe`). If you do not use the template configuration, the `cov-build` command will capture only those Java compile commands that have the same absolute pathname as a compiler that was identified by a preceding `cov-configure` command. Any symbolic links in the pathnames of the configured and executed compile commands will be replaced by real directory names before comparison. For example, in a Unix system, the `inode` of the compile commands must match. Because the pathnames to the Java compile command can vary, you need to configure all of the pathnames in use or to define a template configuration to handle all of them.

JavaScript

Use the configuration, `cov-configure --javascript`.

PHP

Use the configuration, `cov-configure --php`.

Python

Use the configuration, `cov-configure --python`.

Deployment

Because this command invokes the native compiler to determine its built-in macro definitions and the system include directories, you need to run it in an environment that is identical to the one in which your native compiler runs. Otherwise, the emulation will be inaccurate.

Options

--

Indicate the end of `cov-configure` options. Following this option, you can specify additional compiler options. For example, GNU compiler installations that use a non-standard path to the cpp0 preprocessor require the additional GNU -B option to specify its path:

```
> cov-configure --compiler gcc -- -B/home/coverity/gcc-cpp0-location/bin
```

If your build explicitly uses the GNU compiler on the command line with either the `-m32` or `-64` option, also supply the option to the `cov-configure` command. For example:

```
> cov-configure --compiler gcc -- -m32
```

--compiler <name>, -co <name>
Specify the compiler to configure. If the compiler <name> is not in the PATH, specify the full pathname to the compiler. To specify additional compiler options, use `--` followed by the options, for example:

```
> cov-configure --comptype gcc --compiler C:\Mingw\bin\gcc.exe -- -D__STDC__
```

--comptype <type>, -p <type>
Specify the type of compiler to configure. In many cases, `cov-configure` guesses the compiler type based on the `--compiler` argument, but if the name of your compiler is non-standard, specify `--comptype`.

As a general rule, never configure a compiler as a C++ compiler. Do so only in the case of a particular problem that you are trying to work around. If you configure a compiler as a C compiler, `cov-configure` will automatically take care of the C++ case.

To see a full list of supported compiler types, run the `cov-configure --list-compiler-types` option.

For information about configuring compilers, see the *Coverity Analysis 8.0 User and Administrator Guide*.

--delete-compiler-config
This option accepts a compiler configuration by an absolute path or a path relative to the top-level configuration file. Only configurations specified in the top-level configuration will be deleted, otherwise this command has no effect.

Example: Top-level configuration file, `conf/config.xml` contains three configurations:

1. template-gcc-config-0
2. template-msvc-config-0
3. template-javac-config-0

The following example will remove the configuration template-gcc-config-0, leaving the remaining two configurations untouched.

```
cov-configure --delete-compiler-config template-gcc-config-0 -c conf/conf.xml
```

--file-glob <pattern>
[Filesystem capture only] Used in conjunction with `--comptype` to specify a glob pattern to match for source files of the specified type.

For example, the following command creates a configuration for JavaScript that captures only files with a `.js` extension:

```
> cov-configure --comptype javascript --file-glob '*.js'
```

Note that the glob pattern will only match on file names, not on directories or path information.

Do not use the `--file-regex` with this option.

--file-regex <pattern>
[Filesystem capture only] Used in conjunction with `--comptype` to specify a regex pattern to match for source files of the specified type.

For example, the following command creates a configuration for JavaScript that captures only files with a `.js` extension:

```
> cov-configure --comptype javascript --file-regex '^.*\.js$'
```

Note that the regular expression will only match on file names, not on directories or path information.

Do not use the `--file-glob` with this option.

--force
Generate the configuration even if the compiler specified does not behave as expected for a compiler of the specified type.

--javascript
Associates files ending in `.js`, `.html`, and `.htm` with a configuration for JavaScript so they can be saved in the intermediate directory.

--list-compiler-types <output>, -lsct <output>
Lists the supported compiler types described in the `--comptype` option.

--list-compiler-types , -lsct
Generates a list of the supported compiler types. Usage:

```
cov-configure --list-compiler-types
```

--list-configured-compilers <output>, -lscc <output>
Lists the configured compilers defined in your `<install_dir_ca>/config/coverity_config.xml` file. The `output` option defines which output format you want the compiler configuration information displayed. It must one of:

- `csv`
- `json`
- `text`

Each format displays the following categories for each configured compiler:

- Configuration name
- Configuration path ("json" format only)
- Compiler type
- Compiler

- Template configuration
- Enabled options/required arguments

  Template configurations have "Config Args" while instantiated configurations have "Required Args". "Config Args" are used to probe the compiler along with any "Required Args" when instantiating a template configuration.

If the value for any field cannot be determined (for example, if an option is not defined in the configuration file), "null" is printed in that field instead.

The "json" format displays the configuration name AND the full path to the configuration in the `"Config Name"` and `"Config Path` elements. The following example shows the "json" format:

```
{
"Config Name"   : "template-gcc-config-0",
"Config Path"   : "C:\\cygwin\\tmp\\template-gcc-config-0",
"Compiler Type" : "gcc",
"Compiler"      : "gcc",
"Is Template?"  : "yes",
"Config Args"   : "-DBAR"
},
```

The "text" and "csv" formats show only the directory and configuration names (not the full path).

The following example shows the "text" and "csv" formats (they are identical):

```
Config Name, CompType, Compiler, Template?, Config/Required Args
-----------, --------, --------, ---------, --------------------
template-gcc-config-0,gcc,gcc,yes,null
template-gcc-config-1,gcc,g++,yes,null
template-javac-config-0,javac,javac,yes,null
template-java-config-0,java,java,yes,null
template-apt-config-0,apt,apt,yes,null
```

Note that in real usage, $prevent$ and $REAL_CC$ are actual paths.

--list-instrument-vars
: Prints a list of all currently-supported instrumentation variables.

--list-required-arguments, -lsra
: Outputs a list of all the potential required arguments for a given compiler. Pass the `--compiler` or `--comptype` arguments to specify the compiler type on which you want `list-required-arguments` to operate.

--no-header-scan
: Disables performing a header scan for macro candidates during probing of a compiler.

--php
: Configures filesystem capture for PHP source code, by default capturing files with `.php`, `.phtml`, `.php3`, `.php4`, `.php5`, and `.php7` extensions. For supported versions of the PHP language, see

"Language Support" in *Coverity Analysis 8.0 User and Administrator Guide* , and for the complete PHP analysis workflow, see "Getting started with Coverity analyses" in the same guide.

--python
Configures filesystem capture for Python source code, by default capturing files with the `.py` extension. Note that filesystem capture does not detect executable Python scripts without a file extension. For supported versions of the Python language, see "Language Support" in Coverity Analysis User and Administrator Guide.

--set-instrument-var <var_name=var_value>
Sets or overrides an instrumentation variable in the `coverity_config.xml` file. For example:

```
cov-configure --set-instrument-var cov_lib_path=override_lib_path
```

The above command will generate the following xml code in the appropriate section of the compiler configuration file:

```
<instrument_variable>
   <var_name>[[:cov_lib_path:]]</var_name>
   <var_value>override_lib_path</var_value>
</instrument_variable>
```

If the config file(s) already contain a default value for the variable in question, this option will overwrite it with the specified `var_value`.

Note

The `--list-instrument-vars` option provides a list of all supported instrumentation variables, for use with `--set-instrument-var`.

--template, -tm
Provides a template configuration for building with a related set of compilers. The necessary compiler configurations are generated with the required arguments as needed during the build process. For example, if a g++ command that specified -m64 was encountered, a g++ configuration would be generated specifying the -m64 argument.

If you specify this flag, the argument to `--compiler` is a name of the compiler without a path. Do not use the `--version` option with this option.

Note

For the following compilers, you can generate a template configuration without using the `--template` option:

- For GNU GCC and G++ (`gcc` and `g++`):

  ```
  > cov-configure --gcc
  ```

- For Java (`java`, `javac`, `javaw`, and `apt`):

  ```
  > cov-configure --java
  ```

- For Microsoft C/C++ (`cl.exe`):

```
> cov-configure --msvc
```

- For Microsoft C# (`csc.exe`):

```
> cov-configure --cs
```

- For Clang:

```
> cov-configure --clang
```

--template-dir <directory_path>, -td <directory_path>
[Coverity Compiler Integration Toolkit option] Specifies a template directory for custom Coverity Compiler Integration Toolkit templates that override templates found in the default location. This option makes it possible to use custom templates without the need to modify anything in the default template directory. Multiple `--template-dir` options are allowed with directories specified in order of decreasing priority.

--version <version>, -v <version>
For C/C++, specify the compiler version. In many cases, cov-configure will guess the compiler version for you. For Microsoft Visual C/C++, the version is of the form "1310".

For Java, values match valid source levels to the `javac` compiler: '1.4', '1.5', '5', '1.6', '6', '1.7', '7'.

--xml-option <option>
Adds user-specified XML to `coverity_config.xml`. This option is useful for adding items to the file without using an editor.

Usage

```
--xml-option=[tag][@<language>]:value
```

- [tag] is the basic XML tag to be added, for example, `add_arg`.
- [@language] specifies that the switch is only to be added for compiler variants with the given language. Valid values are: `C, C++, Java, CS, ObjC, ObjC++`, or `NC` (where `NC` stands for "Not Compiled" and applies to JavaScript, PHP, and Python). If this specifier is omitted, then the tag will be added for all compiler types being configured.
- *value* is the value contained within the tag. This can be any value, including arbitrary XML.

Simple example:

```
--xml-option=append_arg:-Ihello
```

Simple example in C config only:

```
--xml-option=append_arg@C:-Ihello
```

Arbitrary XML: `--xml-option` allows any number of XML elements. See example below:

```
--xml-option=:<append_arg>-Ihello</append_arg><append_arg> --ppp_translator</
append_arg>
```

Arbitrary XML in C++:

```
--xml-option=@C++:<append_arg>-Ihello</append_arg>
```

C/C++ options

--cygpath <path>
: Specify the path to the directory, which contains the bin directory of the Cygwin installation, if it is not in the PATH environment variable.

--cygwin
: On Windows, indicates that Cygwin is necessary for a GCC compiler. The `cov-configure` command can detect if Cygwin is necessary without this, but you can use this option to force Cygwin if needed.

Shared options

--config <coverity_config.xml> , -c <coverity_config.xml>
: Use the specified configuration file instead of the default configuration file located at <install_dir_sa>/config/coverity_config.xml.

--debug, -g
: Turn on basic debugging output.

--ident
: Display the version of Coverity Analysis and build number.

--info
: Display certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

--redirect stdout|stderr,<filename> -rd stdout|stderr,<filename>
: Redirect either stdout or stderr to <filename>.

--tmpdir <tmp>, -t <tmp>
: Specify the temporary directory to use. On UNIX, the default is $TMPDIR, or /tmp if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>
: Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.

- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

Name

cov-copy-overrun-triage (Deprecated) Migrate triage data from OVERRUN_STATIC and OVERRUN_DYNAMIC defects to OVERRUN.

Synopsis

`cov-copy-overrun-triage` --host <`host`> --port <`port`> --user <`user`> [--password <`password`>] [--triageStoreFilter <`glob`>] [--dryrun]

Description

This command is deprecated as of version 7.0. The `cov-copy-overrun-triage` command copies triage data in a Coverity Connect instance from OVERRUN_STATIC and OVERRUN_DYNAMIC defects to their corresponding OVERRUN defects. OVERRUN defects that already have triage data will not be modified.

All actions performed by this command are recorded in `cov-copy-overrun-triage.log`.

Note that Coverity Connect automatically performs the primary functionality of this command.

Options

--dryrun
: Do not make any modifications on the server. CIDs of interest are listed in `cov-copy-overrun-triage.log`.

--host <`host`>
: The server hostname on which to copy triage data. The server must have a running instance of Coverity Connect.

--password <`password`>
: The password for the user specified by the `--user` parameter. If unspecified, defaults to the value of the environment variable `COVERITY_PASSPHRASE`.

> **Warning**
>
> On multi-user systems, such as Linux, users can see the full command line of all commands that all users execute. For example, if a user uses the `ps -Awf` command, identifying information such as usernames, process identities, dates and times, and full command lines are displayed.

--port <`port`>
: The HTTP port on the Coverity Connect host.

--triageStoreFilter <`glob`>
: Pattern specifying a set of triage stores. For each triage store that matches, triage data will be copied to and from software issues (defects) that are in the same stream.

--user <`user`>
: The user name that is shown in Coverity Connect as having triaged new OVERRUN defects.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

Examples

Copy triage data on host `coverity_server1` for the `Default Triage Store`:

```
> cov-copy-overrun-triage \
  --host coverity_server1 \
  --port 8080 \
  --user admin \
  --password 1256 \
  --triageStoreFilter "Default Triage Store"
```

Copy triage data on host `coverity_server1` for all streams:

```
> COVERITY_PASSPHRASE=1256 \
  cov-copy-overrun-triage \
  --host coverity_server1 \
  --port 8080 \
  --user admin
```

Name

cov-count-lines Perform a line count of the source code.

Synopsis

```
cov-count-lines [--list <file-list>] [--file <file>]
```

Description

Count the number of source code lines in the file(s) specified that are available for Coverity pricing.

Coverity analyzes the number of lines based on the code collected. This may include some third party code which is included in the line count because it is part of the full code that gets compiled.

When counting lines, Coverity strips away blank lines and comments, but does not strip away single braces or parentheses (comments and blank lines are not counted as lines).

Options

--file <filename>
: Count the lines in <filename>. You can use absolute or relative file/path names.

--list <file-list>
: Specify a text file (`<file-list>`), which contains one file name per line. The file names can be absolute or relative, but relative filenames must be relative to the current working directory.

Shared options

--debug, -g
: Turn on basic debugging output.

--ident
: Display the version of Coverity Analysis and build number.

--info
: Display certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

--tmpdir <tmp>, -t <tmp>
: Specify the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>
: Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

Examples

Flag files with a line count greater than 1000:

```
> cov-count-lines --list l | awk '{if ($5 > 1000) print $0;}'
```

Report the line count for the Apache `regcomp.c` file:

```
> cov-count-lines --file /home/user/apache_1.3.33/src/regex/regcomp.c
```

Name

cov-emit Parse and emit a C/C++ source file.

Synopsis

```
cov-emit [OPTIONS] <file.c>
```

Description

The `cov-emit` command parses a source file and outputs it into a directory (emit repository) that can later be analyzed with `cov-analyze`. The `cov-emit` command is typically called by `cov-translate`, which is in turn typically called by `cov-build` (`cov-emit` is a low-level command and is not normally called directly). The `cov-emit` command defines the `__COVERITY__` preprocessor symbol.

Options

--add_type_modifier=[<modifier>]

Enables `cov-emit` to recognize and parse previously unknown type modifiers such as `__data16`, `__code32`, `__huge` etc.

Consider the following code:

```
void foo(int *x) {
  /* Do something */
}

void foo(int __data16 * x) {
  /* Do something else */
}
```

Utilizing the `--add_type_modifier=__data16` will tell `cov-emit` that `__data16` is a type modifier, and it will treat the two functions as distinct. Implicit conversions between types is done with reliance on the native compiler to enforce the conversion rules - that is, it is assumed that all conversions are valid.

Note

You can specify one type modifier to be the *default*. The *default* setting is used when instantiating templates to mimic the native compiler's usage of default type modifiers as shown in the following example:

```
--add_type_modifier=__data8,__data16:default
```

--allow-incompat-return-types

Allows a prototype of a function to specify different return types than those in the actual function definition.

--allow_incompat_throw

Indicates to `cov-emit` that it should not report an error if there are multiple prototypes of the same function with incompatible C++ exception specifications.

--angle_include_search_first={default|user|system}
Controls the order that directories are searched when trying to find an included file. This option applies to files added by `#include <...>`.

- `default` - Indicates that there is no change from previous behavior.
- `user` - Indicates that user include directories (added with `-I`) will be searched first.
- `system` - system include directories (added with `--sys_include`) will be searched first.

The `--sys_include_first` option, which is now depreciated, is equivalent to `--angle_include_search_first=system`.

--arg_file <file>
Read arguments from the response file `<file>`. This is typically done by `cov-translate` to avoid command-line length limitations.

The response file format is as follows:

```
<compiler_args>
  <cov_emit_cmd_args>
    <arg>[cov-emit arg]</arg>
    <arg>[another cov-emit arg]</arg>
  </cov_emit_cmd_args>
</compiler_args>
```

Note

Note that spaces within the `<arg>` tags are interpreted literally. This means that `<arg>-DFOO=bar</arg>` will work, while `<arg> -DFOO=bar</arg>` will cause an error, as the argument is interpreted as a source file name.

--c
Compile standard C code (C89).

--c++11
Enable c++11 language features.

--c++14
Enable c++14 language features.

--c++
Compile standard C++ code. This is the default.

--c99
Enable C 99 extensions to the C programming language.

--cache_include_search
If you use large numbers of #include search directories with the `-I` option, specify this option to speed up compilation.

--char_bit_size <integer>
Used to specify the size (in bits) of the char type. If this option is not specified, the default is 8-bit chars.

--coverity_has_extension <key>[=<integer>]
Controls the `__has_extension()` preprocessor macro, setting the value to `TRUE` (or the optional `<integer>`) for any defined routines. For example, `--coverity_has_extension c99` results in `__has_extension(c99) == TRUE`.

The `no_` prefix sets the value to `FALSE` (i.e. `--no_coverity_has_extension c99 # __has_extension(c99) = FALSE`).

--coverity_has_feature <key>[=<integer>]
Controls the `__has_feature()` preprocessor macro, setting the value to `TRUE` (or the optional `<integer>`) for any defined routines. For example, `--coverity_has_feature c99` results in `__has_feature(c99) == TRUE`.

The `no_` prefix sets the value to `FALSE` (i.e. `--no_coverity_has_feature c99 # __has_feature(c99) = FALSE`).

--coverity_has_intrinsic <key>[=<integer>]
Controls the `__has_intrinsic()` preprocessor macro, setting the value to `TRUE` (or the optional `<integer>`) for any defined routines. For example, `--coverity_has_intrinsic __builtin_asin` results in `__has_intrinsic(__builtin_asin) == TRUE`.

The `no_` prefix sets the value to `FALSE` (i.e. `--no_coverity_has_intrinsic __builtin_asin # __has_intrinsic(__builtin_asin) = FALSE`).

--coverity_option <key>[=<integer>]
Controls the `__option()` preprocessor macro, setting the value to `TRUE` (or the optional `<integer>`) for any defined routines. For example, `--coverity_option gnuversion` results in `__option(gnuversion) == TRUE`.

The `no_` prefix sets the value to `FALSE` (i.e. `--no_coverity_option gnuversion # __option(gnuversion) = FALSE`).

--cygwin
This switch tells `cov-emit` to attempt to convert Unix-based (Cygwin) paths into their corresponding Windows (real) paths.

-D <identifier>[=<value>]
Add a macro definition of `<identifier>` with optional `<value>`.

--dir
Specifies the emit repository (an intermediate directory) into which the `cov-emit` command outputs its results.

-E
Only preprocess the source file.

--enable_128bit_int
The following switches provide the ability to turn on/off 128-bit integer types, independent of `gnu_version`:

- `--enable_128bit_int`
- `--no_enable_128bit_int`
- `--enable_128bit_int_extensions`
- `--no_enable_128bit_int_extensions`

`--enable_128bit_int` implies `--enable_128bit_int_extensions`, however the same does not apply to the `no_*` variants.

`--enable_128bit_int` enables type `__int128`, while `--enable_128bit_int_extensions` enables types `__int128_t` and `__uint128_t`.

--encoding <enc>
Specifies the encoding of source files. Use this option when the source code contains non-ASCII characters so that Coverity Connect can display the code correctly. The default value is US-ASCII. Valid values are the ICU-supported encoding names:

US-ASCII

UTF-8

UTF-16

UTF-16BE
UTF-16 Big-Endian

UTF-16LE
UTF-16 Little-Endian

UTF-32

UTF-32BE
UTF-32 Big-Endian

UTF-32LE
UTF-32 Little-Endian

ISO-8859-1
Western European (Latin-1)

ISO-8859-2
Central European

ISO-8859-3
Maltese, Esperanto

ISO-8859-4
North European

ISO-8859-5
Cyrillic

ISO-8859-6
Arabic

ISO-8859-7
Greek

ISO-8859-8
Hebrew

ISO-8859-9
Turkish

ISO-8859-10
Nordic

ISO-8859-13
Baltic Rim

ISO-8859-15
Latin-9

Shift_JIS
Japanese

EUC-JP
Japanese

Note

EUC-JP is now a valid output object encoding. See --output_object_encoding.

ISO-2022-JP
Japanese

GB2312
Chinese (EUC-CN)

ISO-2022-CN
Simplified Chinese

Big5
Traditional Chinese

EUC-TW
Taiwanese

EUC-KR
: Korean

ISO-2022-KR
: Korean

KOI8-R
: Russian

windows-1251
: Windows Cyrillic

windows-1252
: Windows Latin-1

windows-1256
: Windows Arabic

Note

If your code is in SHIFT-JIS or EUC-JP, you must specify the `--output_object_encoding SHIFT-JIS` or `--output_object_encoding EUC-JP` option (respectively) in order to avoid receiving STRING_OVERFLOW false positives.

For more information, see --output_object_encoding.

--encoding_selector <encoding>/<regular expression>
: Treats all files with file names that match the given, case-sensitive regular expression as though they have the specified encoding. For a list of valid encodings, see the `--encoding` option to this command. The regular expression syntax is a Perl regular expression, as described in http://perldoc.perl.org/perlre.html.

Encoding selectors also apply to files that are included in source files, not just to the files specified on the `cov-emit` command line. This behavior allows for a finer granularity in selecting encodings.

Note that encoding selectors have a higher priority than the `--encoding` option. If the `cov-emit` command line contains both `--encoding <encoding>` and `--encoding_selector <encoding>/<regular expression>`, and the regular expression is a match for the file that is currently getting opened, the encoding specified through the encoding selector will take precedence.

--encoding_selector_case <encoding>/<regular expression>
: Identical to `--encoding_selector`, except that the regular expression is case insensitive.

--error_limit <number>
: For a file that fails to compile, specifies the number of errors that are output to `build-log.txt` before moving to the next file. Default is 5.

--float_bit_patterns
: Enables parsing the ARM Development Studio C/C++ language extension that allows floating-point bit pattern literals, for example:

```
float f = 0f_00000000;
double d = 0d_0000000000000000;
```

--force
Disables incremental compilation by forcing `cov-emit` to compile and generate output for a file, even if a copy of that file has already been compiled and is present in the Intermediate Directory.

--gcc
Allow parsing of C code with gcc (GNU) extensions.

--g++
Allow parsing of C++ code with g++ (GNU) extensions.

--gnu_version=<version>
Specifies the version of the GNU compiler to emulate. Only required when the code you are compiling exploits version-dependent features or bugs in the `gcc` or g++ compilers. If the version of `gcc` you are using is 3.4.2, for example, then specify the version as 30402.

--has_extension_macro
Enables the intrinsic `__has_extension()` preprocessor macro.

--has_feature_macro
Enables the intrinsic `__has_feature()` preprocessor macro.

--has_intrinsic_macro
Enables the intrinsic `__has_intrinsic()` preprocessor macro.

--ignore_std
Specifies that the namespace "std" should be ignored entirely. g++ versions prior to 3.0 ignored the "std" namespace.

--incompat_proto
Allows a prototype of a function to specify a different set of arguments than those in the actual function definition.

-I <dir>
Add a directory to search for `#include` files. Directories added with this switch are considered 'user' include directories.

The default behaviour is to search for headers in the order that both the `-I` and `--sys_include` were specified on the command line, regardless of `#include` type. This can be adjusted using --angle_include_search_first and --quote_include_search_first.

--lazy_hex_pp_number
This option affects grammar in which a statement such as '0x1e-1' can be parsed as either a single pp-number (C++11 2.10 [lex.ppnumber]) or as a subtraction expression.

When using the Coverity Compiler Integration Toolkit, you will only need to use this option if the compiler correctly parses the example above, but `cov-emit` does not.

In order to determine if you need to use this option, you will receive an error message from `cov-emit` that looks like the following:

```
"test.cpp", line 3: error: extra text after expected end of number
int foo = (0xD8E-0xD64);
               ^
```

--lowercase_header_filenames
: In the cov-emit preprocessor, when the source refers to a header filename, turn it into all-lowercase before asking the operating system for the file. This can be useful when transitioning to a case-sensitive filesystem. Also translates backslash to slash, and removes a leading drive letter and (back)slash, as these are needed in the same situations.

--macro_stack_pragmas
: Enables parsing of GNU macro stack manipulation pragmas (`#pragma push_macro` and `#pragma pop_macro`). This option is enabled by default in most cases, but may be automatically disabled in certain compatibility modes.

  To manually disable this option, use `--no_macro_stack_pragmas`.

--microsoft
: Allow parsing of Microsoft extensions.

--multiline_string
: Allow the Coverity compiler to accept multi-line strings. Multi-line strings are supported by compilers such as gcc 3.4.

--nested_comments
: Allow the Coverity compiler to accept nested block comments. Nested comments are supported by compilers such as Renesas RX 2.03.

--new_array_args
: Enables parsing for options to array element constructors invoked by the `new[]` variant of the C++ memory allocation operator.

--no_atomic_commit
: This option is deprecated as of the 5.0 release.

--no_caa_info
: Do not collect the information required for Architecture Analysis in the intermediate directory.

--no_exceptions
: Disable parsing for exception handling in C++.

--no-headers
: This option is deprecated as of the 5.0 release.

--no_macro_stack_pragmas
: Disables parsing of GNU macro stack manipulation pragmas (`#pragma push_macro` and `#pragma pop_macro`). This option is enabled by default in most cases, but may be automatically disabled in certain compatibility modes.

  To enable parsing of these pragmas, use `--macro_stack_pragmas`.

--no_predefined_feature_test_macros
Do not predefine the testing macros described in `--predefined_feature_test_macros`.

This is the default behavior of `cov-emit`.

--no_predefined_stdc
Do not predefine `__STDC__`.

--no_predefines
Do not predefine any macros internally. All macro definitions must be in the source code or explicitly on the command line.

This is the default behavior for `cov-emit`. Use the `--predefines` option to predefine specific macros.

Note that `--no_predefines` has no effect on the following macros, which may still be predefined even if this option is specified:

- Certain C/C++ standard macros (e.g., `__FILE__, __LINE__`)
- Macros that begin with "`__COVERITY`"
- Macros that are controlled by another switch (e.g., `__STDC__` and `--no_predefined_stdc`)

--old_g++
For GNU versions prior to 3.x, specifies a more permissive version of g++ compatibility.

--option_macro
Enables the intrinsic `__option()` preprocessor macro.

--output_object_encoding
Specifies the output character encoding. This option accepts an encoding as a required argument. The accepted encodings are `Shift-JIS` and `EUC-JP`. For example:

```
--output_object_encoding EUC-JP
```

Using `--shiftjis_encode_obj` is effectively the same as specifying `--output_object_encoding Shift-JIS`.

If `--output_object_encoding` is not specified, then the object encoding is UTF-8.

--ppp_translator <translator>
Add --ppp_translator `<translator>` to the `cov-emit` command line to translate files before they are preprocessed. Possible `<translator>` values are:

- `cmd:<command>`– Pipes file through `<command>` .
- `replace/<from>/<to>`– Replaces regular expression `<from>` with `<to>` . The regular expression syntax is a Perl regular expression, as described in http://perldoc.perl.org/perlre.html. The '/' character can be replaced with any other character; and this separator character can be quoted with a backslash '\'.

--pp_sizeof
Allows the use of `sizeof()` in preprocessing directives. When compiling, the argument to `sizeof` can be anything permitted in an expression. However, when preprocessing, it is only possible to use built-in types like `int`. This means that preprocessing output might be different than what the compiler encounters during compilation. This feature is nonstandard and only supported by a few compilers.

--pre_preinclude <file.h>
Specify header file that should be processed prior all other source and header files.

--predefined_feature_test_macros
Compatible with C++ only. The WG21 working paper N3694 provides guidelines for predefined feature testing macros. When this option is specified, `cov-emit` will predefine the appropriate macros as suggested by these guidelines.

--predefines
When specified, `cov-emit` will predefine additional macros based on the current emulation mode.

For example:

```
> cov-emit --microsoft --predefines test.c
```

The above command will predefine `_MSC_VER`, while the following command predefines `__GNUC__`.

```
> cov-emit --gcc --predefines test.c
```

Note that `--predefines` has no effect on the following macros:

- Certain C/C++ standard macros (e.g., `__FILE__, __LINE__`)
- Macros that begin with "`__COVERITY`"
- Macros that are controlled by another switch (e.g., `__STDC__` and `--no_predefined_stdc`)

--preinclude_macros <file.h>
Specify macros-only header file that should be processed immediately after the files specified with `--pre_preinclude` option (see above) and prior to all other source and header files.

--preinclude <file.h>
Specify header file that should be processed immediately after the files specified with `--pre_preinclude` and `--preinclude_macros` options (see above) and prior to all other source and header files.

--print_predefined_macros
Print all predefined macros (and their values) to stdout.

--ptrdiff_t_type <builtin-type>
Specify the type of ptrdiff_t. This is stored in the `__COVERITY_PTR_DIFF_TYPE__` macro (as the type, not the character code). If unspecified, `__COVERITY_PTR_DIFF_TYPE__` is set to the same type as ptrdiff_t (for example, "signed int"). The character code for <builtin-type> is typically one of the signed types shown in --size_t_type. For example, `--ptrdiff_t_type i` sets the type of ptrdiff_t to "signed int".

--quote_include_search_first={default|user|system}
Controls the order that directories are searched when trying to find an included file. This option applies to files added by `#include "..."`.

- `default` - Indicates that there is no change from previous behavior.
- `user` - Indicates that user include directories (added with `-I`) will be searched first.
- `system` - system include directories (added with `--sys_include`) will be searched first.

--short_enums
Enables optimization of enumeration sizes. The size of each enumeration will be set based on the largest values present when this option is specified.

--size_t_type <builtin-type>
Specify type of size_t. This is stored in the `__COVERITY_SIZE_TYPE__` macro (as the type, not the character code). If unspecified, `__COVERITY_SIZE_TYPE__` is set to the same type as size_t (generally, the default is "unsigned int"). Use an unsigned integral type from the single-character codes for `<builtin-type>` as follows:

```
          a  # signed char
          h  # unsigned char
          s  # short
          t  # unsigned short
          i  # int
          j  # unsigned int
          l  # long
          m  # unsigned long
          x  # long long, __int64
          y  # unsigned long long, __int64
```

For example, "`--size_t_type j`" sets the type of size_t to "unsigned int".

--sys_include <dir>
Add a directory to search for `#include` files. Directories added with this switch are considered 'system' include directories.

The default behaviour is to search for headers in the order that both `-I` and `--sys_include` were specified on the command line, regardless of `#include` type. This can be adjusted using --angle_include_search_first and --quote_include_search_first.

--system_encoding <enc>
Specifies the encoding to use when interpreting command line arguments and file names. If not specified, a default system encoding is determined based on host OS configuration.

See --encoding for a list of accepted encoding names.

--type_alignments <builtin-type>
Specify type of type_alignments. The <builtin-type> string consists of the ABI chars shown in --size_t_type, plus the following:

```
          f  # float
```

```
          d  # double
          e  # long double, __float80
          P  # Coverity extension: pointer
```

and lengths. For example, `--type_alignments x8li4s2P4` sets type_alignments to `long long 8, long & int 4, short 2, ptr 4`.

--type_sizes <builtin-type>
Specify type of type_sizes. The <builtin-type> string consists of the ABI chars shown in --size_t_type, plus the following:

```
          w  # wchar_t
          f  # float
          d  # double
          e  # long double, __float80
          n  # __init128
          o  # unsigned __init128
          g  # __float128
          P  # Coverity extension: pointer
```

and lengths. For example, "`--type_sizes w4x8li4s2P4`", sets type_sizes to "wide char 4 bytes, long long 8, long & int 4, short 2, ptr 4".

If unspecified, `cov-emit` uses the machine's native type sizes.

The C standard mandates that `sizeof(char) == 1` and `sizeof(<any other type>) == <multiple of sizeof(char)>`. Therefore, all type sizes should be specified as multiples of a `char` size (and `char` should always be size 1). To set the bit size of a `char`, see `--char_bit_size`.

For example, assume you have a compiler that has the following:

- `16-bit chars`
- `16-bit shorts`
- `32-bit ints`
- `32-bit longs`

The correct arguments for this compiler are:

```
cov-emit --char_bit_size 16 --type_sizes st1ijlm2
```

Note

Note that if this option specifies contradictory sizes for signed and unsigned versions of the same type, the last value specified will be used. For example, `--type_sizes i4j6` will set the length of "int" and "unsigned int" to 6, and the 4 will be ignored.

--wchar_t_keyword
Indicates that `cov-emit` should treat the type wchar_t as a keyword built into the language.

--wchar_t_name <identifier>
Uses the specified identifier for the `wchar_t` intrinsic type. This option does not imply `--wchar_t_keyword`.

--wchar_t_type <builtin-type>
Specifies the type `wchar_t`, where `<builtin-type>` is one of the unsigned integral types shown in --size_t_type. For example, "`--wchar_t_type j`" sets the type of wchar_t to "unsigned int".

-U <identifier>
Undefine the macro `<identifier>` .

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

See Also

cov-build

cov-translate

Name

cov-emit-cs Parse C# source code and emit output to the intermediate directory.

Synopsis

```
cov-emit-cs --dir <intermediate_directory> [OPTIONS] [sourcefiles]
```

Description

This command parses source code, decompiles referenced assemblies, and saves CSC compiler output to the emit repository in the intermediate directory.

Options

--addmodule <file>
: Identifies a module that is referenced by the compilation but not added to the emit repository. The location of the module can be absolute or relative. For rules on specifying the location, see --reference.

  It is an error if the referenced module is not found. To change this behavior, see `--allow-missing-refs`.

  You must specify this option separately for each module.

--allow-missing-refs
: Issues a warning if any referenced assemblies are missing. If you do not set this option, missing assemblies result in an error that stops the process. This error applies to explicitly referenced assemblies (see --reference and --addmodule) that are absolute or not found in the Common Language Runtime (CLR) system directory. The error also applies to `mscorlib.dll` (see --nostdlib). Note that missing `csc.rsp` assemblies (see --noconfig) always result in a warning.

--codepage <codepage>
: Identifies the numeric codepage corresponding to codepages that are supported by CSC with the `/codepage` option. Source file encodings are determined in the following manner:

  If a byte order mark (BOM) is present in the source file, the command uses the BOM-related encoding. If a BOM is *not* present, encoding is determined in the following manner:

  - The character encoding of the specified codepage is used. If a codepage is not specified, the command attempts to detect and use UTF-8. If neither of the preceding alternatives is possible, the command uses the system default codepage.

--compiler-dir <directory>
: Identifies the CLR system directory. It is an error to specify a directory that does not exist. The CLR system directory is used as a search path for referenced assemblies (see `--reference`) and to locate the `csc.rsp` file (see `--noconfig`). If `--compiler-dir` is not specified, the command defaults to `$SYSTEM_ROOT/Microsoft.NET/Framework/<version>`, where <version> is the latest supported framework version (for details, see the *Coverity 8.0 Installation and Deployment Guide*). It is an error if no suitable CLR system directory is found.

--define <define>
Corresponds to the CSC preprocessor directive and `/define` option. Note that it is necessary to specify a separate `--define` option for each directive.

--dir <intermediate_dir>
Identifies the intermediate directory into which this command emits source files and referenced assemblies. An error occurs if the specified intermediate directory exists but is not valid, or if the directory does not exist and cannot be created.

This option is required.

--enable-cs-parse-error-recovery
Makes `cov-emit-cs` fall back to error recovery mode when compilation errors are encountered during the processing of source files.

This option is disabled by default.

--force
By default, if all the specified source files exist in the emit repository with the same timestamp as the file on disk, the command skips the files and exits with a successful return code. Specifying the `--force` option makes the command process the source files, even if they exist unchanged in the emit repository.

--lib <directory>, -L <directory>
Identifies a library directory to use when searching for referenced assemblies (see `--reference`). A warning (not an error) occurs if you specify a directory that does not exist. This option corresponds to the CSC `/lib` option.

You must specify this option separately for each library directory.

--noconfig
Ignores the `csc.rsp` file under the CLR system directory (see `--compiler-dir`). If this option is not set, the references `/r` or `/reference` within `csc.rsp` are added to the list of referenced assemblies. Any `csc.rsp` references that are not absolute filenames are subject to the search directory rules (for details, see `--reference`). Corresponds to the CSC option `/noconfig`.

--nostdlib
Disables the default behavior of searching for `mscorlib.dll` in the CLR system directory and adding the file to the list of referenced assemblies. If that file is not found, the next search in this directory is for a `csc.exe.config` file that specifies a `requiredRuntime` version. If a version is found, the search continues to the corresponding directory (the parent directory of the CLR system directory).

This option corresponds to the CSC `/nostdlib` option.

--no-friends
Prevents the compilation from accessing internal types or members. This option works by disabling the processing of compiler output retrieved through the `--out` option. This behavior corresponds to CSC behavior in the case where `/out` is not specified and a default name is used.

--no-banner
Suppresses the `cov-emit-cs` application name and version banner.

--out <file>
Specifies the compiler output file, which is then used by the command to access internal types or members in referenced assemblies.

Subsequent calls to `cov-emit-cs` will not re-emit the output file if a referenced assembly is found.

It is necessary to specify `--out` or `--no-out` (or the alternative, `-n`), else an error will occur.

--no-out, -n
Indicates that there is no compiler output.

It is necessary to specify `--out` or `--no-out` (or the alternative, `-n`), else an error will occur.

--reference <[alias=]filename> | -r <[alias=]filename>
Identifies an assembly to provide for compilation and addition to the emit repository, unless this setting is overridden by other options. The location of the assembly can be absolute or relative. If relative, the command searches the following paths in the following order:

1. Current working directory.
2. CLR system directory.
3. Each directory specified by the `--lib` option, in the order specified.

By default, it is an error if a referenced file cannot be found on disk. See `--allow-missing-refs` for complete rules.

Note that `[alias=]` identifies an extern alias directive, just as it does for CSC.

Example with an alias:

```
--reference v1=my.dll
```

--unsafe
Allows analysis C# constructs that are declared with the `unsafe` modifier. This option corresponds to the `/unsafe` option to CSC.

--use-link-semantics
Corresponds to the `/link` option to CSC, which changes how the compiler treats certain COM `interop` types.

Any unrecognized options result in an error, which causes an immediate exit with an appropriate error message.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.

- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

Name

cov-emit-java Parses one or more source files and outputs it into an intermediate directory.

Synopsis

```
cov-emit-java [OPTIONS] [SOURCE FILES]
```

Description

The `cov-emit-java` command parses Java source code and bytecode, and saves `javac` outputs. It stores these outputs to an emit repository for subsequent static analysis and outputs it into a directory (emit repository) that can later be analyzed with `cov-analyze`. The `cov-emit-java` command is typically called by `cov-build`.

You need to invoke this command when you are running Java Web Application Security analyses and in the rare case that you cannot compile your Java code with `cov-build`. For details about the latter case, see the discussion of the alternative build process in the Coverity Analysis.

When specifying multiple source files, you need to separate each source file by a space, for example:

```
src/pkg/SomeClass.java src/pkg/OtherClass.java
```

Note that you can specify the options to `cov-emit-java` in any order, but the list of source files must appear last.

Error codes

0
: Success. Also applies to errors from which it is possible to recover.

  The `cov-emit-java` command is less strict than `javac` in that it recovers from some invalid or inconsistent inputs by discarding problematic code, printing a message, and continuing to function.

2
: Most likely, a user-generated error, such as code that will not compile.

4
: Irrecoverable error. Please contact `support@coverity.com` if you receive this error code.

Options

--android-apk <Android_APK_file>
: [Used for Coverity Extend SDK checkers that analyze Android applications] Identifies an Android APK file that is associated with a specified input file and read by custom Extend SDK checkers that are built for Android analyses.

  Requirement: When using this option, you must also specify an `--input-file` option to this command. For information about custom Android checker development, see the section, "Reporting events and defects on input files" in *Coverity Extend SDK 8.0 Checker Development Guide*.

--auto <project_directory>
Allows you to specify a directory where source (*.java), input Jar files (*.jar), and compiler outputs (*.class) can be found. If you set this option, cov-emit-java will recursively search for these items.

Note that --auto <project_directory> is functionally equivalent to the following:

```
--findsource <project_directory>
--findjars <project_directory>
--compiler-outputs <project_directory>
```

If necessary, you can specify --findsource, --findjars, or --compiler-outputs options along with the --auto option. This sort of specification might be useful if you have a simple project that requires specific Ant or JDK dependencies.

Example:

```
cov-emit-java --dir <intermediate_directory> --auto $PROJECT_ROOT --
findjars $ANT_HOME:$JAVA_HOME
```

--bootclasspath <directories_or_jar_files>
Allows you to specify a list of directories or Jar files, which must be separated by a semi-colon (;) on the Windows platform, and by a colon (:) on other platforms. Classes specified through the --bootclasspath are emitted, but the bodies of their methods are not, because cov-emit-java expects to have models for them. Coverity Analysis for Java comes with models for the entire Java Runtime Environment (JRE) and Android SDK, which should address all cases.

Like javac, cov-emit-java searches these entries for bytecode with the referenced classes when attempting to resolve names in source files. The directories/jar files are searched in the order specified in <directories_or_jar_files>.

Generally, you do not need to specify this option because cov-emit-java selects the bootclasspath of the JRE that comes with Coverity Analysis for Java. However, if you are compiling code against a non-standard JRE, one that is not API-compatible with the standard JRE, then you might need to use this option.

--classpath <directories_or_jar_files>
Allows you to specify a list of directories or Jar files, which must be separated by a semi-colon (;) on the Windows platform, and by a colon (:) on other platforms. Wildcard (*) in classpath is supported. The wildcard will find all Jar files in the given directory (ie: foo/* finds all jars in foo/). Like javac, cov-emit-java searches these entries for bytecode with the referenced classes when attempting to resolve names in source files. The directories/jar files are searched in the order specified in <directories_or_jar_files>. Since Coverity Analysis analyzes these class files, it is better, when possible, to specify the implementations that are loaded at runtime rather than the stubs that are used for compilation.

Note

Note that directories specified with --classpath will only be searched for Jar files if the wildcard is used. Otherwise, directories will be searched only for bytecode.

If a directory (`foo/`) may contain bytecode *as well as* Jar files, you should include both "`foo/`" AND "`foo/*`" in your arguments to `--classpath`.

The `cov-emit-java` command captures the same bytecode in Jar files referenced on the classpath that `java` or `javac` find (that is, bytecode where the package name matches the directory within the Jar file).

The `cov-emit-java` command respects the `Class-Path` entry in Jar manifests.

If the `--classpath` option is not specified, the current directory will be used as the default classpath.

The `cov-emit-java` command is not affected by the CLASSPATH environment variable.

Note

When loading certain jar files, in particular the Scala runtime library, `cov-emit-java` can consume more than 10GB of memory. This can cause out-of-memory failures on low-memory systems, including 32-bit systems in general.

--compiler-outputs `<class_files_or_directories>`

Captures a list of class files that have debug symbols for subsequent analysis by FindBugs™. For proper display of defects detected by the FindBugs analysis, such class files should have been compiled with all debugging information (for example, with `javac -g`).

The list contains class files separated by the classpath separator (colon or semi-colon). Any directories in the list will be recursively searched for class files. Compiler outputs should be specified on the same command line as the source code from which they were compiled. Subsequent invocations of `cov-emit-java` on the same source files will replace the compiler outputs from the previous invocation with those of the new invocation.

Jar files passed to `--compiler-outputs` will have their contained class files included. Directories passed to `--compiler-outputs` will not have their contained Jar files included.

Note

Do not pass obfuscated bytecode to this option.

--dir `<intermediate_directory>`

Specifies the emit repository (an intermediate directory) into which the `cov-emit-java` command outputs its results.

This option is required.

The command fails with an error if either of the following are true:

- The specified directory exists but is not a valid intermediate directory.
- The intermediate directory does not exist and cannot be created.

--enable-java-parse-error-recovery
Enables the error recovery algorithm that produces the highest emit percentage in most cases. Currently, this option enables per-class error recovery, but its behavior might change in future.

Explicitly enabling an error recovery algorithm on the command line will automatically disable any incompatible error recovery algorithms.

--enable-java-per-class-error-recovery
This option can help to increase the percentage of source files that can be emitted by attempting to use the class files of the source files that cause parse errors. It could potentially increase the time to emit files, but is usually faster than `--enable-java-per-file-error-recovery` because it does not try to emit each file one at a time.

Per-class error recovery is unlikely to correct `cov-emit-java` crashes. It also requires the presence of output class files. To address such issues, see `--enable-java-per-file-error-recovery`.

This option cannot be used with `--enable-java-per-file-error-recovery`.

--enable-java-per-file-error-recovery
Use this option if your native Java compiler is able to compile your source code successfully, but `cov-emit-java` crashes. This option might also help when `cov-emit-java` has parse errors and there are no class files available for per-class error recovery.

--encoding <`character_encoding`>
Applies the specified character encoding to all source files processed by this invocation of `cov-emit-java`. Defaults to UTF-8. This default might not be the same one that `javac` uses.

Supports the following encodings (note that these differ from what `javac` supports):

US-ASCII

UTF-8

UTF-16

UTF-16BE
UTF-16 Big-Endian

UTF-16LE
UTF-16 Little-Endian

UTF-32

UTF-32BE
UTF-32 Big-Endian

UTF-32LE
UTF-32 Little-Endian

ISO-8859-1
Western European (Latin-1)

ISO-8859-2
Central European

ISO-8859-3
Maltese, Esperanto

ISO-8859-4
North European

ISO-8859-5
Cyrillic

ISO-8859-6
Arabic

ISO-8859-7
Greek

ISO-8859-8
Hebrew

ISO-8859-9
Turkish

ISO-8859-10
Nordic

ISO-8859-13
Baltic Rim

ISO-8859-15
Latin-9

Shift_JIS
Japanese

EUC-JP
Japanese

Note

EUC-JP is now a valid output object encoding. See --output_object_encoding.

ISO-2022-JP
Japanese

GB2312
Chinese (EUC-CN)

ISO-2022-CN
Simplified Chinese

Big5
: Traditional Chinese

EUC-TW
: Taiwanese

EUC-KR
: Korean

ISO-2022-KR
: Korean

KOI8-R
: Russian

windows-1251
: Windows Cyrillic

windows-1252
: Windows Latin-1

windows-1256
: Windows Arabic

MacRoman
: The `cov-emit-java` treats MacRoman as Macintosh.

Note

Some other unsupported encoding names might be supported if a known alias is supported. For example, the Java canonical x-EUC-TW is mapped to EUC-TW.

The `cov-emit-java` command attempts to tolerate encoding errors and logs a warning when it finds bytes that cannot be decoded.

--findears <directory_list>
: [Java Web application option] Searches the specified directories recursively for EAR (Enterprise Archive) files and adds the ones that it finds to the emit in the intermediate directory. The directory list must be separated by a semi-colon (;) on the Windows platform, and by a colon (:) on other platforms.

In most cases, Coverity recommends that you use only one of these related options (`--findwars`, `--findwars-unpacked`, `--findears`, or `--findears-unpacked`). The option should match the final packaged `web-app` format. Otherwise, the search might find and emit unwanted temporary build artifacts.

Note

The Web application archives should not contain obfuscated classes.

--findears-unpacked <directory_list>
[Java Web application option] Searches the specified directories recursively for unpacked `web-app` root directories and add the ones that it finds to the emit in the intermediate directory. The `web-app` root directories are identified by the presence of a `META-INF/application.xml` file. The directory list must be separated by a semi-colon (;) on the Windows platform, and by a colon (:) on other platforms.

In most cases, Coverity recommends that you use only one of these related options (`--findwars`, `--findwars-unpacked`, `--findears`, or `--findears-unpacked`). The option should match the final packaged `web-app` format. Otherwise, the search might find and emit unwanted temporary build artifacts.

Note

The Web application directories should not contain obfuscated classes.

--findjars `<Jar_containing_directories>`
[Java Web application option] Allows you to specify a list of directories, which must be separated by a semi-colon (`;`) on the Windows platform, and by a colon (`:`) on other platforms. The `cov-emit-java` command searches these directories recursively for Jar files and adds the ones that it finds to the classpath. The directories are searched in the order specified in `<Jar_containing_directories>`.

Note that this option can result in an error if the number of Jar files exceeds the limit on the number of open files that is allowed by your operating system.

--findsource `<source_directories>`
Lists directories, which must be separated by a semi-colon (`;`) on the Windows platform, and by a colon (`:`) on other platforms. The `cov-emit-java` command searches these directories recursively for source files. It process the source files that it finds as if they were specified directly on the `cov-emit-java` command line.

--findwars <directory_list>
[Java Web application option] Searches the specified directories recursively for WAR (Web application archive) files and adds the ones that it finds to the emit in the intermediate directory. The directory list must be separated by a semi-colon (;) on the Windows platform, and by a colon (:) on other platforms.

In most cases, Coverity recommends that you use only one of these related options (`--findwars`, `--findwars-unpacked`, `--findears`, or `--findears-unpacked`). The option should match the final packaged `web-app` format. Otherwise, the search might find and emit unwanted temporary build artifacts.

Note

The Web application archives should not contain obfuscated classes.

See also --webapp-archive and --findwars-unpacked.

--findwars-unpacked <directory_list>
[Java Web application option] Searches the specified directories recursively for unpacked `web-app` root directories and adds the ones that it finds to the emit. The `web-app` root directories are identified by the presence of a `WEB-INF/web.xml` file. The directory list must be separated by a semi-colon (;) on the Windows platform, and by a colon (:) on other platforms.

In most cases, Coverity recommends that you use only one of these related options (`--findwars`, `--findwars-unpacked`, `--findears`, or `--findears-unpacked`). The option should match the final packaged `web-app` format. Otherwise, the search might find and emit unwanted temporary build artifacts.

Note

The Web application directories should not contain obfuscated classes.

See also --webapp-archive and --findwars.

--force
Disables incremental compilation by forcing `cov-emit-java` to compile and generate output for a file, even if a copy of that file has already been compiled and is present in the Intermediate Directory.

--help
Prints a usage message to the command console and exits.

--ignore-sccs
Ignores all directories named `SCCS`. This is useful for version control systems that store metadata with `.java`, `.jar`, and `.class` extensions in directories named `SCCS`.

--input-file <resource_file>
[Used for Coverity Extend SDK checkers that analyze Android applications] Identifies a resource file, typically a `AndroidManifest.xml` file, that can be read by custom Extend SDK checkers built for Android analyses. This option can be specified multiple times on the command line.

Requirement: When using this option, you must also specify the `--android-apk` option. For information about custom Android checker development, see the section, "Reporting events and defects on input files" in *Coverity Extend SDK 8.0 Checker Development Guide*.

--javac-version
Identifies which implementation's bugs to emulate. Oracle Javac is the standard, but there are places where Oracle Javac does not conform to the specification and EDGjfe does. To accommodate this, EDGjfe implements Oracle's bugs and ties it to the `--javac-version` switch. If `--source` is explicitly set, then `--javac-version` is implicitly set to the same value. If `--javac-version` is explicitly set, then `--source` is implicitly set to the same value. If both are explicitly set, then both have the value they are explicitly set to.

--jvm-max-mem
[Java Web Application Security option] Sets the value of the the JVM that is used for invoking the Jasper engine for JSP compilation. The option accepts an integer that specifies a number of megabytes (MB). The default value is 1024.

--minimal-classpath-emit
Limits the group of emitted JAR files to those needed for compilation of the Java files. The default behavior without this option is to emit all the JAR files in the classpath regardless of whether they are referenced by a Java file in the compilation. This option can improve performance of Java builds with large numbers of unused JAR files on the classpath at the risk of not capturing all the dependencies of the those JAR files. For example if `A.java` references `A.jar`, which has dependencies on `B.jar`, this option will prevent `B.jar` from getting emitted even if `B.jar` is on the classpath.

--no-compiler-outputs
Indicates that the `--compiler-outputs` option is intentionally unspecified. Use of this option is not recommended because both the dynamic analysis for Java Web application security and the FindBugs analysis rely on a compiler output specification. Without emitting compiler outputs, you can expect to see false positive XSS reports and missing FindBugs reports.

It is an error to run `cov-emit-java` without exactly one of the following options: `--no-compiler-outputs` or `--compiler-outputs`.

--skip-war-sanity-check
[Java Web application option] Suppresses a failure in the case that the emit process determines that expected contents of the Web application (web-app) archives are missing.

This option overrides the following sanity check on each WAR file or Web application directory on the command line:

- The check that each contains a `/WEB-INF` directory and `/WEB-INF/web.xml` file.

This option overrides the following sanity check on the set of all Web application archives or directories on the same command line:

- The check that the Web applications do not contain enough (>20%) of the classes captured during build capture or manual `cov-emit-java` invocations.

These checks are designed to catch cases where someone passes the wrong items to `--webapp-archive`. Turn off this check *only if* you are certain that you are passing the correct Web application files or directories to `cov-emit-java`, despite the warnings.

For additional details, see --webapp-archive and --skip-webapp-sanity-check.

--source `<Java_version>`
Identifies which version of the Java language to emulate. For example, `--source=1.7` will allow `cov-emit-java` to handle binary literals and other features that appeared in Java 7. If `--source` is explicitly set, then `--javac-version` is implicitly set to the same value. If `--javac-version` is explicitly set, then `--source` is implicitly set to the same value. If both are explicitly set, then both have the value they are explicitly set to..

--sourcepath `<source_directories>`
Lists directories, which must be separated by a semi-colon (`;`) on the Windows platform, and by a colon (`:`) on other platforms. Like `javac`, the `cov-emit-java` command searches these directories for source files that contain referenced classes. If no `--sourcepath` is provided, the sourcepath will default to the expanded classpath.

--webapp-archive <archive_file_or_dir>, --war <archive_file_or_dir>, --ear <archive_file_or_dir>
[Java Web application option] The `--webapp-archive` and `--war` options store the contents of the specified Web Archive (WAR, `.war`) file, Enterprise Archive (EAR, `.ear`), or directory with the unpacked contents of either to the intermediate directory (emit repository) and prepares them for analysis. For these two options, the `cov-emit-java` command inspects the file or directory that is provided as argument and it guesses its type, based on the presence of WEB-INF (for WAR) or META-INF (for EAR), falling back to WAR by default. The `--ear` is similar, but it only interprets its argument as an EAR.

These contents include files with the following extensions: `.dtd`, `.css`, `.htm`, `.html`, `.jar`, `.js`, `jsp`, `.jspf`, `.jspx`, `.properties`, `.tag`, `.tagf`, `.tagx`, `.tld`, `.txt`, `.wsdl`, `.xml`, and `.xsd`.

These options can be passed multiple times to store and analyze multiple archives.

Use of this option is required as a precondition to performing analysis on Java Enterprise Edition (Java EE), servlet-based Web applications, regardless of whether `cov-build` is used. (The `cov-build` command does not specifically recognize Web application archives as this option does.)

Important!

You need to emit the JSP files so that the analysis can find and report issues it finds in them. If these files are not present in the WAR file, false negatives will occur, particularly in XSS defect reports.

It is also important to emit the original JSP source, even in cases where the build normally pre-compiles the JSP files into classes and packages those into the WAR file.

Lastly, the Web application archive or directory should not contain obfuscated classes.

Example:

```
cov-emit-java --dir my/intermediate/dir
  --webapp-archive path/to/webapp.war
  --webapp-archive path/to/webapp2.war
```

You can also specify a list of directories to search for WAR or EAR files (or unpacked directories) using one of the following options to this command: `--findwars`, `--findwars-unpacked`, `--findears`, or `--findears-unpacked`.

After using this option, you can run an analysis with the `--webapp-security` option to `cov-analyze`. See "Running a security analysis on a Java Web application".

See also --findwars and --findwars-unpacked.

--verbose
Outputs extra information about inputs. Valid values: 0, 1, 2, 3, 4. Defaults to 1.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

Examples

```
> cov-emit-java --findsource src --findjars lib:build-lib/ --dir
  my/intermediate/dir --compiler-outputs build/classes/:build/junitclasses/
```

See Also

cov-build

cov-analyze

Name

cov-export-cva Produce a Coverity CVA file to be used by Architecture Analysis.

Synopsis

```
cov-export-cva --dir <intermediate_directory> --output-file <filename>
```

Description

The `cov-export-cva` command produces a CVA file to be used by Architecture Analysis. This allows you to create and retrieve the CVA file from your intermediate directory instead of having to run a commit and then log in to Coverity Connect.

A `cov-analyze` command with at least one `--strip-path` option must be run at some point prior to running this command. After this command is executed, the filename you specified in `--output-file <filename>` is created.

The file is compressed in `gzip` format and is ready to be read by Architecture Analysis.

Options

--dir <intermediate_directory>
: Pathname to an intermediate directory that is used to store the results of the build and analysis.

--output-file <filename>
: The name of the file CVA file to be created.

--output-tag <name>
: Use this option if you used it when generating analysis results. See the --output-tag option to `cov-analyze`.

Shared options

--debug, -g
: Turn on basic debugging output.

--ident
: Display the version of Coverity Analysis and build number.

--info
: Display certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

--tmpdir <tmp>, -t <tmp>
: Specify the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

See Also

cov-analyze

Name

cov-find-function Find a mangled or internal function name when given part of the actual name.

Synopsis

```
cov-find-function [OPTIONS] --dir <intermediate_directory> <name>
```

Description

In the Extend SDK, it is relatively easy to match on a C function with a specific name by passing the name to the `Fun` pattern constructor. Otherwise, the mangled name must be used to disambiguate which overloaded function should be matched. The `cov-find-function` command looks for all of the mangled names that contain the given `<name>`.

For Java and C#, this command finds the internal representation used by the analysis.

This command makes it easier to find the name needed for matching on a specific function or method, even an overloaded one. The `cov-find-function` command can accept a regular expression to denote a set of functions to display.

Options

--cpp
: Filters the results by the C/C++ programming language. See also, `--cs` and `--java`.

--cs
: Filters the results by the C# programming language. See also, `--cpp` and `--java`.

--dir <intermediate_directory>
: Pathname to an intermediate directory that is used to store the results of the build and analysis.

--exact, -x
: Assume that `<name>` is a full function prototype, not just a substring of the mangled name. Not commonly used.

--include-builtins
: Look for given functions in the built-in model library.

--java
: Filters the results by the Java programming language. See also, `--cs` and `--cpp`.

--model-file <file.xmldb>
: Look for the function in the given user model file. See also, the --model-file option to `cov-analyze`.

--module <module>, -m <module>
: Requires `--show`. Pick the model for module `<module>` when showing a model for a function. Values can be *`all`*, *`generic`*, *`security`*, *`concurrency`*, *`symbian`*, *`stack_use`*, *`uninit`*, and *`ptr_arith`*. Defaults to *`generic`*.

--output <directory>, -of <directory>
: Specify the directory in which the output of `--save` is stored. The default is the current directory. If the directory does not exist it is created.

--save
: Save the model file in `<key>.<module>.models.xml`, a description of edges in `<key>.<module>.model_edges`, and a .ps file of the graph (as shown by `--show`) in `<key>.<module>.ps`. Each model is uniquely identified by an MD5 hex-key. `cov-find-function` uses this *key* to immediately find a model. If given a function name, it does a linear search of the model database. This search might take some time, but when it finds a function, it prints its model key for the specified module.

  Requires the `dot` command from the Graphviz package.

--show, -s
: Show the model for the function. Requires `dot` (from the Graphviz package) as well as `ggv` (GNOME's PS viewer).

--subdir
: When used in conjunction with `--use-emit`, specifies a subdirectory in which to look for the given function. Might substantially speed execution.

--use-emit, -ue
: Iterate over the emit directory to find functions, instead of looking at the cache database. Useful if the cache is not present (for example, it has been cleaned or the analysis has never been run) or corrupted.

--user-model-file <file.xmldb>
: [Deprecated] This option is deprecated as of version 7.7.0. Use --model-file instead.

Shared options

--debug, -g
: Turn on basic debugging output.

--ident
: Display the version of Coverity Analysis and build number.

--info
: Display certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

--tmpdir <tmp>, -t <tmp>
: Specify the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

Example

Look for a function named `get` and save the results in the directory `get_models`:

```
> /cov-find-function --dir /home/user/apache_intdir  --info --save --output
 get_models get
```

Name

cov-format-errors Generate static HTML pages of defect reports.

Synopsis

`cov-format-errors`

```
    --dir <intermediate_dir>
    [--emacs-style]
    [--exclude-files <regex>]
    [--file <file_substring>]
    [--filesort]
    [--function <function>]
    [--functionsort]
    [--html-output <directory>]
    [--include-files <regex>]
    [--output-tag <name>]
    [--security-file <file>]
    [--title <text>]
    [-x]
    [-X | --noX]
```

Description

The `cov-format-errors` command reads defects from an intermediate directory and creates static HTML pages in the specified directory.

Deprecated behavior: By default, this command writes HTML output into the `<intermediate_directory>/output/errors` directory, but this usage is deprecated. Instead, you should use the `--html-output` option to specify the HTML output directory.

Note

`cov-format-errors` works for Java, C/C++, and C#.

To commit defects to Coverity Connect, use cov-commit-defects instead of `cov-format-errors`.

Note that `cov-format-errors` does not have access to any triage information stored in Coverity Connect. Therefore, the output of `cov-format-errors` will include defects even though they have been marked as *Intentional* or *False Positive* within Coverity Connect. In addition, the output of `cov-format-errors` is only accessible to users who have access to the local file system; it is not made available through a network service.

Options

--dir <intermediate_directory>
: Pathname to an intermediate directory that is used to store the results of the build and analysis.

--emacs-style
: Print a short version of the defect event with line numbers to `stdout`, formatted as gcc compiler warnings. This option is useful for integration into an editor such as Emacs.

--exclude-files <regex>
Do not output defects from files that are in paths that match the specified regular expression. You cannot use this option multiple times in the same command line. You can use it with the `--include-files` option. If you use both options together, `--exclude-files` takes precedence over `--include-files`. For example, defects from a given file are excluded from the output in the case that the regular expressions both include and exclude the file.

Example that excludes paths that contain both `/bar/` and `.c`:

```
--exclude-files '/bar/.*\.c$'
```

The example excludes the following file:

`/foo/bar/test.c`

The example does not exclude the following files:

`/foo/test.c`

`/foo/bar/test.cc`

Note

The example above uses single quotes around the string value. However, your command interpreter might require double quotes (for example, `"/bar/.*\.c$"`).

--file <filename>
Only generate pages for errors in files containing `<filename>` as a substring of the full pathname.

--filesort
Sort rows by filename.

--function <func>
Only generate pages for errors in function `<func>`.

--functionsort
Sort rows by function name.

--html-output <directory>
Write the HTML results to the specified directory, and create this directory first if it does not exist.

You must either specify a directory previously written by `cov-format-errors`, or a directory that is empty, or that does not yet exist.

--include-files <regex>
Output defects only from files that are in paths that match the specified regular expression. You cannot use this option multiple times in the same command line. You can use it with the `--exclude-files` option. If you use both options together, `--exclude-files` takes precedence over `--include-files`. For example, defects from a given file are excluded from the output in the case that the regular expressions both include and exclude the file.

Example that includes `/foo/`:

```
--include-files '/foo/'
```

The example includes the following file:

`/bar/foo/test.c`

The example excludes the following file:

`/bar/test.c`

Note

The example above uses single quotes around the string value. However, your command interpreter might require double quotes (for example, `"/foo/"`).

--json-output-v1 <filename>, --json-output-v2 <filename>
Writes `cov-format-errors` output to the specified file in JSON format.

`--json-output-v2` is the recommended JSON output option because it contains the most complete set of information. `json-output-v1` is supported for backward compatibility.

--lang <language>
Write event messages in the specified language. Currently, the supported values are `en` (for English) and `ja` (for Japanese). The default language is English (`en`).

--misra-only
[Deprecated in 8.0] Using this option will result in an error.

--noX
Do not build cross-reference information. Normally, cross-reference information is built if the `-x` option is specified.

--output-tag <name>
Use this option if you used it when generating analysis results. See the --output-tag option to `cov-analyze`.

--security-file <license file>, -sf <license file>
Path to a Coverity Analysis license file. If not specified, this path is given by the `security_file` element in the XML configuration file, or `license.dat` in the same directory as `<install_dir_sa>/bin`.

--strip-path <directory>
Strips the prefix of a file name path in error messages and references to your source files. If you specify the `--strip-path` option multiple times, you strip all of the prefixes from the file names, in the order in which you specify the `--strip-path` argument values.

The leading portion of the path is omitted if it matches a value specified by the this option. For example, if the actual full pathname of a file is `/foo/bar/baz.c`, and `--strip-path /foo` is specified, then the name attribute for the file becomes `/bar/baz.c`.

--title <title>
Specify a title for the generated index pages.

-X
Run the `cov-internal-build-xrefs` command first. Without this option, the identifiers in the source code will not be hyperlinked. When this has been done once on an intermediate directory, it does not need to be done again until the intermediate data changes. `-x` automatically implies `-X` unless `--noX` is also specified.

-x
Use cross-reference information when building static pages. Without this flag, the identifiers in the source code will not be hyperlinked. This option needs to be specified every time you want the generated pages to have cross-reference information.

Shared options

--debug, -g
Turn on basic debugging output.

--ident
Display the version of Coverity Analysis and build number.

--info
Display certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

--tmpdir <tmp>, -t <tmp>
Specify the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>
Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

See Also

cov-commit-defects

Name

cov-generate-hostid Return host id information for node-locked licensing.

Synopsis

`cov-generate-hostid` [-of `<filename>`]

Description

The `cov-generate-hostid` command outputs the host id information needed for node-lock licensing. Email this information to `license@coverity.com`.

Options

--output-file `<filename>`, -of `<filename>`
The file to create with this information. By default, the information is sent only to standard out.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

Example

Create the host id information in the named file:

```
    > cov-generate-hostid -of /user/foo/cov_host.txt
```

Name

cov-help Display help for a command.

Synopsis

`cov-help <command>`

Description

The `cov-help` command displays help for a specified command.

You can also read each command's help page by specifying the `--help` option on the command line. For example:

```
> cov-analyze --help
```

Environment Variables

$PAGER
: The program through which to pipe output. The `$PAGER` variable is ignored if the output is not a terminal. If `$PAGER` is not set, the `cov-help` command looks for the `less` or `more` commands in paths set by the `$PATH` environment variable.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

Name

cov-import-msvsca Import Microsoft Visual Studio Code Analysis issues

Synopsis

```
cov-import-msvsca --dir <intermediate_directory> <MSVSCA_xml_files> [--
append] [--skip-unrecognized] [--no-threshold-check]
```

Description

`cov-import-msvsca` allows you to import Microsoft Visual Studio Code Analysis (MSVSCA) issues on C# code into your Coverity Analysis intermediate directory by specifying XML files generated by Microsoft Visual Studio Code analysis (MSVSCA). The results can then be committed to and viewed in Coverity Connect. MSVSCA is integrated into many versions of Microsoft Visual Studio, including Professional, Premium, and Ultimate editions of Visual Studio 2012.

Also known as FxCop, Code Analysis for Managed Code reports issues in programs compiled for the .NET Framework, including languages other than C#. Although `cov-import-msvsca` can import most issues from non-C# managed code, `cov-import-msvsca` works best for C# code and always imports results into the C# domain of an intermediate directory. `cov-import-msvsca` does not currently support importing results from Code Analysis for C/C++ or Code Analysis for Drivers. `cov-import-msvsca` requires a valid license for Coverity Analysis for C#.

Event descriptions imported by `cov-import-msvsca` are not localized in Coverity Connect. However, you can import localized results (including localized event descriptions). If you configure Visual Studio for a particular localization, the Microsoft Visual Studio Code analysis (MSVCA) results will use that localization. You can then import the localized results to Coverity Connect. Users will see the localized even descriptions in Coverity Connect regardless of their Coverity Connect language configuration.

The `cov-import-msvsca` command is only available on the Windows platform.

For Microsoft's complete list of Code Analysis warnings for Managed Code, see http://msdn.microsoft.com/en-us/library/dd380629.

Importing MSVSCA results and viewing them in Coverity Connect

1. Run the MSVSCA analysis to generate one or more XML files of MSVSCA results.

   When the Code Analysis is run from Visual Studio or as part of a build, an XML file with the results for each project is saved in the same directory as the compiler output. For example:

   ```
   <path_to_compiler_output>.CodeAnalysisLog.xml
   ```

   The `FxCopCmd.exe` program that is included with MSVSCA can also generate results in the expected XML format.

2. Run `cov-import-msvsca`, referencing the XML file(s) with MSVSCA results.

   For example:

   ```
   cov-import-msvsca --dir idir <xml_files> ...
   ```

The `cov-import-msvsca` command expects that the assemblies analyzed to generate the MSVSCA XML files were compiled with debug information, and that the assembly, its debug symbol (pdb) file, and original source code have not moved on the filesystem.

Imported MSVSCA results and Coverity Analysis results can be combined in the same intermediate directory using `--append`. All C# results in an intermediate directory will be committed to a single stream in Coverity Connect. If the intermediate directory only contains imported results, source files in the relevant project that do not contain defects might not be stored and committed. (It is possible to correct this with a supplementary `cov-import-results --append` with just source files.)

`cov-import-msvsca` tracks how many MSVSCA issues are in the XML input files and what proportion of those are dropped because it is unable to associate them with a source code line. This can occur if there is missing debug information and/or missing source files. If that proportion is greater than 10%, it is reported as a user error (see --no-threshold-check below) and no issues are imported.

Only results that are generated by the Code Analysis with Visual Studio 2012 are officially supported, but other versions are compatible. A warning is printed if an unsupported input file is specified, but the tool attempts to import anyway.

Note

When `cov-import-msvsca` runs on high-density files (files with more than 100 issues that also average more than 1 issue for every 10 lines of code), the console will print a warning that names all the files that exceed the threshold, and the import process will exclude all issues associated with the affected files from the intermediate directory. This change prevents the Coverity Connect source browser from becoming too crowded with issues.

To suppress this density check (allowing all issues to be imported) in version 7.0, define the environment variable COVERITY_ALLOW_DENSE_ISSUES (COVERITY_ALLOW_DENSE_ISSUES=1) when running the commands.

3. Commit the results to Coverity Connect.

   Commit the results with `cov-commit-defects`, specifying the intermediate directory to which you imported the MSVSCA results.

   When you (or another user) log into Coverity Connect the MSVSCA issues are similar to other Coverity Analysis issues, but the checker name has a prefix of `MS.`. The rest of the checker name uses the Code Analysis ID. For example, `MS.CA1303`.

Options

--dir <intermediate_directory>
: Pathname to an intermediate directory that is used to store the results of the build and analysis.

--append
: Specify that issues should be appended to the intermediate directory. Without `--append`, existing C# issues are deleted before storing the imported issues.

See also, --output-tag.

--codepage <identifier>
Specifies Microsoft code page source encoding. <identifier> is an integer represents the code page identifier, for example `--codepage 1201`. For the list of code page identifiers, see http://msdn.microsoft.com/en-us/library/windows/desktop/dd317756%28v=vs.85%29.aspx.

Source with a BOM will have the encoding auto-detected, even if `--codepage` or `--encoding` is specified. However, if you specify `--encoding`, `cov-import-msvsca` will log a warning recommending that you use `--codepage` instead. `--codepage` results in using the .NET mechanisms for decoding, which more closely mimic the Microsoft tools.

You cannot use `--codepage` and `--encoding` together.

--encoding <encoding_name>
Specify that source files are read using the named character encoding, such as `UTF-8`. The default is based on detection of byte-order marks and falling back on the operating environment default character encoding. For a list of encoding options, see --encoding <enc> from `cov-emit`.

--no-threshold-check
By default, if more than 10% of reported issues do not have file and line information because their referenced assemblies, their associated pdbs, or referenced source files (either from the MSVSCA XML file(s) or assembly pdb) are missing, then `cov-import-msvsca` will fail without importing any issues and print an informative error message that mentions `--no-threshold-check`. However, if `--no-threshold-check` is specified, these messages are just informative and are not treated as an error. The defects that depend on the missing information are omitted from the results and the remaining issues are imported normally.

--output-tag <name>
Specifies a non-default location within the intermediate directory for the results of one or more imports. The name can be anything you choose, using characters allowed in file names. When specified *without* the --append option, prior results found in this location are replaced. When specified *with* `--append`, new results are added to the result set.

@@<response_file>
Specify a "response file" that contains a list of additional command line arguments, such as a list of input files. Each line in the file is treated as one argument, regardless of spaces, quotes, etc. The file is read using the platform default character encoding. Using a response file is recommended when the list of input XML files is long or automatically generated.

--skip-unrecognized
By default, `cov-import-msvsca` fails if a specified input file is not in a recognized format or if the list of input files is empty. If `--skip-unrecognized` is specified, files in an unrecognized format are simply skipped with a warning, and the list of files can be empty. Thus, the translation of any input files in the recognized format proceeds normally, even if there are none.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

See Also

cov-commit-defects

Name

cov-import-results Import third-party issues into Coverity Connect

Synopsis

```
cov-import-results --dir <intermediate_directory> [--append] <filename>
```

Description

`cov-import-results` is the command tool for the Coverity Connect Coverity Third Party Integration Toolkit. It imports to the intermediate directory source code files of any type and any issues pertaining to those source files generated by third-party analysis tools.

This command imports third party issue information through a specified JSON import file (<filename> in the above syntax).

Note

When `cov-import-results` runs on high-density files (files with more than 100 issues that also average more than 1 issue for every 10 lines of code), the console will print a warning that names all the files that exceed the threshold, and the import process will exclude all issues associated with the affected files from the intermediate directory. This change prevents the Coverity Connect source browser from becoming too crowded with issues.

To suppress this density check (allowing all issues to be imported) in version 7.0, define the environment variable COVERITY_ALLOW_DENSE_ISSUES (COVERITY_ALLOW_DENSE_ISSUES=1) when running the commands.

See Using the Coverity Third Party Integration Toolkit for information about using the Coverity Third Party Integration Toolkit and creating the JSON import file.

Options

--append
: Append issues to any issues that exist in the intermediate directory. If `--append` is absent, all of the issues in the intermediate directory are deleted before importing. If `--append` is present, they are not deleted.

  See also, --output-tag, --cpp, --java, --cs, --other-domain.

--cpp
: Makes the command import only C/C++ source files and issues. This option is the default. You can only import results for one language domain (C/C++, C#, Java, or Other) domain per invocation of `cov-import-results`. However, you can use `--append` to add results from other language domains before committing the results to Coverity Connect.

  See also, --output-tag, --append, --java, --cs, --other-domain.

--cs
: Makes the command import only C# source files and issues. You can only import results for one language domain (C/C++, C#, Java, or Other) domain per invocation of `cov-import-results`.

However, you can use `--append` to add results from other language domains before committing the results to Coverity Connect.

See also, --output-tag, --append, --cpp, --java, --other-domain.

--dir <intermediate_directory>
Pathname to an intermediate directory that is used to store the results of the build and analysis.

--java
Makes the command import only Java source files and issues. You can only import results for one language domain (C/C++, C#, Java, or Other) domain per invocation of `cov-import-results`. However, you can use `--append` to add results from other language domains before committing the results to Coverity Connect.

See also, --output-tag, --append, --cpp, --cs, --other-domain.

--other-domain
Makes the command impoprt source files and issues for languages other than C/C++, Java, and C#. You can only import results for one language domain (C/C++, C#, Java, or Other) per invocation of `cov-import-results`. However, you can use `--append` to add results from other language domains.

See also, --output-tag, --append, --cpp, --cs, --java.

--output-tag <name>
Specifies a non-default location within the intermediate directory for the results of one or more imports. The name can be anything you choose, using characters allowed in file names. When specified *without* the `--append` option, prior results found in this location are replaced. When specified *with* `--append`, new results are added to the result set.

--no-banner
Hide the version of Coverity Analysis and build number.

--strip-path <path>, -s <path>
Strips the prefix of a file name path in error messages and references to your source files. If you specify the `--strip-path` option multiple times, you strip all of the prefixes from the file names, in the order in which you specify the `--strip-path` argument values.

This option is also available with `cov-commit-defects` and `cov-analyze`.

The leading portion of the path is omitted if it matches a value specified by the this option. For example, if the actual full pathname of a file is `/foo/bar/baz.c`, and `--strip-path /foo` is specified, then the name attribute for the file becomes `/bar/baz.c`.

Note

Coverity recommends using this option for a number of reasons:

You can enhance end-to-end performance of the path stripping process by using this option during the analysis of your source code, rather than when committing the analysis results to Coverity Connect.

It shortens paths that Coverity Connect displays. It also allows your deployment to be more portable if you need to move it to a new machine in the future.

Shared options

--debug, -g
Turn on basic debugging output.

--ident
Display the version of Coverity Analysis and build number.

--info
Display certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

--tmpdir <tmp>, -t <tmp>
Specify the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

Name

cov-link Create an intermediate directory with resolved duplicate function calls for C/C++.

Synopsis

```
cov-link {--collect | <link_file>...} [--compile-arg <arg> | --compile-arg-
regex <regex> | --no-compile-arg <arg> | --no-compile-arg-regex <regex> | --
source-file-regex <source_file_regex>]... {{--output-dir <output_dir> | --
output-file <output_file>} --dir <intermediate_dir>} [OPTIONS]
```

Description

Sometimes the same file is compiled several times with different command-line options. To avoid errors in function-call resolution (especially in C code, which does not have name mangling), you can use the `cov-link` command to create a new intermediate directory that contains a subset of the files that are in your original intermediate directory. This new intermediate directory can then be analyzed without function-call resolution issues.

The input consists of an intermediate directory (with an emit repository), plus a set of translation units. The translation units are either collected dynamically from the emit repository with the `--collect` option, or they are specified inside one or more of the link files that were previously created with `cov-link`.

Typically, `cov-link` is first called with the `--collect` option to produce a link file. You can look at this file for clues on which filters to apply to generate an intermediate directory with just the subset of files that you are interested in. Once you have an idea of which filters you need to apply, you can call `cov-link` again with your filters to produce a new intermediate directory. This new intermediate directory can then be analyzed.

To use the `cov-link` command:

1. Run `cov-link` with the `--collect` and `--output-file` options. This operation collects linkage information on all files compiled in an emit directory.
2. Create one or more additional link files by filtering information using either an argument or a portion of the pathname that was used during command-line compilation. Compiled files are identified based on:
   - A portion of the pathname to the file when it was compiled. Use the `--source-file-regex` option to specify a Perl regular expression to use when looking at the pathname.
   - The options given on the command line when it was compiled. Use the `--compile-arg`, `--compile-arg-regex`, `--no-compile-arg`, and `--no-compile-arg-regex` options to group by command-line options.
3. Use the link files created in the previous steps, and the emit repository in the original intermediate directory, to create a new intermediate directory with an emit repository with resolved function calls.
4. Use `cov-analyze` on the new intermediate directory.

For more information, including detailed examples, see the *Coverity Analysis 8.0 User and Administrator Guide*.

Options

Input options:

You must specify one of the following options. You must not use both together.

--collect, -co
: Collects linkage information from all of the entries in an emit repository.

`<link_file>`
: Specifies which source files are linked together. You can specify multiple link files.

Filter options:

These options are not required but can be specified multiple times.

--compile-arg `<arg>`, -a `<arg>`
: Specify an argument that was given when compiling the files on the command line.

--compile-arg-regex `<regex>`, -r `<regex>`
: Specify an argument that was given when compiling the specified files, as a Perl regular expression.

--no-compile-arg `<arg>`, -na `<arg>`
: Specify an argument that was NOT given when compiling the files on the command line.

--no-compile-arg-regex `<regex>`, -nr `<regex>`
: Specify an argument that does NOT match any argument given when compiling the files on the command line, as a Perl regular expression.

--source-file-regex `<source_file_regex>`, -s `<source_file_regex>`
: Specify a portion of the source file pathname that was used during compilation, as a Perl regular expression. You can use a forward slash (`/`) as a directory separator in this string, for example `/proj1/` matches if `proj1` is a directory that is in the pathname. Note that on Windows, the matching is case-insensitive, and (`/`) is used as the directory separator (not `\`). You can specify this option more than once (as in `-s <source_file_regex> -s <source_file_regex>`). If there are several `-s` options, the source file's name only needs to match one of the specified expressions.

Output options:

You must specify one of the following options. You must not use both together.

--output-dir `<output_dir>`, -odir `<output-dir>`
: Specifies an intermediate directory for the `cov-link` command to create. If you use this option, you must also use the `--dir` option to this command.

  Note that the `--dir` option to the `cov-analyze` command will use the specified `<output_dir>` as its value.

--output-file `<output-file>`, -of `<output-file>`
: Specifies the pathname to the link file that is created. If you use `--collect`, any existing file with this name is replaced. If you specify `--source-file-regex`, any existing file with this name is appended to.

If you use this option, you must also use the `--dir` option to this command.

Shared options

--dir <intermediate_directory>
Pathname to an intermediate directory that is used to store the results of the build and analysis.

--ident
Display the version of Coverity Analysis and build number.

--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>
Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

Examples

Create a link file based on an existing emit repository:

```
> cov-link --dir . --collect -of /usr/foo/all.link
```

Create a link file based source files with `apache` in the pathname:

```
> cov-link --dir . -s  /apache/ -of /usr/foo/link_reports/apache.link  \
    all.link
```

Create a link file based on the source files with `apache_1.3.33` in the pathname, include only the files that were compiled with the `DEBUG` macro defined on the command line, and then create an intermediate directory with an emit repository:

```
> cov-link --dir . -a -DDEBUG  -s /apache_1.3.33/   \
    -of /usr/foo/link_reports/apache1333_DEBUG.link  all.link
> cov-link  --dir .  --output-dir emit_apache1333_DEBUG \
    /usr/foo/link_reports/apache1333_DEBUG.link
```

Create a new emit repository based on the source files with `apache_1.3.33` in the pathname, and include only the files that were compiled with the `DEBUG` macro defined on the command line (same as previous example, but without creating the intermediary link file):

```
> cov-link --dir . --collect -a -DDEBUG -s /apache_1.3.33/   \
    --output-dir emit_apache1333_DEBUG
```

See Also

cov-analyze

Name

cov-make-library Create a user model file from C/C++, Objective-C, Objective-C++, C#, or Java source code.

Synopsis

```
cov-make-library [-of <modelfile>] [--security] [--concurrency] [--enable
CHECKER] [--disable CHECKER] [--disable-default] [--disable-webapp-security]
[--make-dc-config] [--quality] [--webapp-security] [-co <compiler>] [--
compiler-opt <compiler_option>] <sources>
```

Description

The `cov-make-library` command creates user model files from source files. User model files contain information that overrides what `cov-analyze` can derive for itself. See the *Coverity 8.0 Checker Reference* for more information about how to model source files. There are also examples in the `<install_dir>/library` directory.

Note

The files in the `<install_dir>/library` directory should not be provided as arguments to `cov-make-library`. You should instead create your own new files for models. Using the existing files creates duplicate, identical user models and Coverity default models.

The file is appended if it already exists, and created if it does not exist. The search order used to determine where to create the model file is the filename specified by `-of` or a default value of `<install_dir>/config/user_models.xmldb`.

The `cov-make-library` command works by calling `cov-emit` to parse and emit the source files, followed by `cov-analyze`, and then `cov-collect-models` to collect the analyzed models.

Default behavior
: For C/C++, Objective-C, and Objective-C++, the default behavior of this command is to generate models for checkers that are enabled by default. For Java and C#, the default behavior of this command is to generate a model for use by all checkers, quality and security. Some of the command options allow you to limit the generation of models to those used by groups of checkers.

Source files are compiled as C, C++, Objective-C, Objective-C++, C#, or Java depending on their file extension:

- Compiled as C code: `.c` extension
- Compiled as Objective-C code: `.m` extension
- Compiled as C++ code: `.cc`, `.CC`, `.cp`, `.cpp`, `.cxx`, `.c++`, and *with the exception of Windows*, `.C` extensions
- Compiled as Objective-C++ code: `.mm` extension
- Compiled as C# code: `.cs` extension

- Compiled as Java code: `.java` extension

Options

--classpath <directories_or_jar_files>
: [Java option] Lists directories or Jar files, which must be separated by a semi-colon (`;`) on the Windows platform, and by a colon (`:`) on other platforms. Like `javac`, `cov-make-library` searches these entries for bytecode with the referenced classes when attempting to resolve names in source files. See also the `--classpath` option to `cov-emit-java`. For `cov-make-library`, it is possible to use stubs.

--compiler <compiler>, -co <compiler>
: [C/C++ option] Specifies a previously configured compiler (configured with `cov-configure --compiler`) that is used to determine how to compile the files (using `cov-translate`). This is useful if you need to include standard headers. For example:

```
> cov-configure --compiler ABC
> cov-make-library --compiler ABC foo.c
```

--compiler-opt <opt>
: [C/C++ option] Specify an option to the compiler specified by `--compiler` (or `cov-emit` is no compiler was specified). For instance, you can specify an include directory with `--compiler-opt -I --compiler-opt include_dir`

--concurrency
: [C/C++ option] Use this option if you write a custom model using concurrency primitives.

--disable <CHECKER>
: [C/C++ option. Deprecated for C# and Java.] Disables the creation of function models used by the specified checker.

  Note that this option disables a checker only if it is enabled by default *and not enabled in any other way*, for example, through a group enablement option such as `--all`.

--disable-default
: [C/C++, C#, Java option] Disables the creation of function models for *all* checkers. Use options such as `--concurrency`, `--security`, `--quality`, and `--webapp-security` to choose a set of checkers for which to generate models.

  When using this option, you must also use an enablement option, such as `--quality` or `--security`. It is an error to use `--disable-default` without such an option.

--disable-webapp-security
: [Java, C# option] Disables the creation of models for Web Application Security checkers (for example, XSS and SQLI). You typically use this option when you only want to generate models for Java quality checkers. For a complete list of Web Application Security checkers, see the "Coverity Checkers" table in the *Coverity 8.0 Checker Reference*.

  See also, `--webapp-security` and `--quality`.

--enable <CHECKER>
[C/C++ option. Deprecated for C# and Java.] Enables the creation of function models used by the specified checkers. If you want the preview checkers to check the source that uses your custom models, you must enable those checkers with this option.

--java
[Deprecated] Deprecated in version 7.0 because the command automatically detemines the language based on the Java file extension.

--make-dc-config
[C/C++ only] Upgrades models for the deprecated SECURE_CODING checker to DC.CUSTOM_CHECKER checker configurations. Specifically, the option searches for `__coverity_secure_coding_function__` models in the source code and generates a JSON configuration file for a custom checker called DC.CUSTOM_CHECKER. The configuration file specifies function names and information found in the custom models.

Example:

```
> cov-make-library -of config.json --make-dc-config my_models.c
```

To use the resulting configuration file in the analysis, you simply pass it through the `--dc-config` option.

Example:

```
> cov-analyze --dir <intermediate_dir> --dc-config config.json -en
 DC.CUSTOM_CHECKER
```

The `--make-dc-config` option is also available to `cov-collect-models`.

--output-file <modelfile>, -of <modelfile>
Specify the name of the output model file. The default file name is `user_models.xmldb`, at `<install_dir_sa>/config/`.

--quality
[C/C++, C#, Java option] Generates models for quality checkers, including concurrency checkers. Use this option with `--disable-default` to generate models for only the quality checkers. For a list of quality checkers, see the "Coverity Checkers" table in the *Coverity 8.0 Checker Reference*.

See also, `--disable-default`, `--disable-webapp-security`, and `--webapp-security`.

--reference <referenced_assembly>
[C# option] Specify a referenced assembly.

--security
[C/C++ option] Use this option if you write a custom model using security-related checkers such as TAINTED_DATA, TAINTED_STRING, STRING_SIZE, and STRING_NULL.

--security-file <license file>, -sf <license file>
Path to a valid Coverity Analysis license file. If not specified, this path is given by the security_file tag in the Coverity configuration, or `.security` in the same directory as `cov-analyze`. A valid license file is required to run the analysis.

--webapp-security
[Web application security option] Generates models for Web Application Security checkers (for example, XSS and SQLI). Use this option with `--disable-default` to generate models for only the Web Application Security checkers. For a complete list of Web Application Security checkers, see the "Coverity Checkers" table in the *Coverity 8.0 Checker Reference*.

See also, `--disable-default`, `--disable-webapp-security`, and `--quality`.

Shared options

--config `<coverity_config.xml>` , -c `<coverity_config.xml>`
Use the specified configuration file instead of the default configuration file located at `<install_dir_sa>/config/coverity_config.xml`.

--debug, -g
Turn on basic debugging output.

--ident
Display the version of Coverity Analysis and build number.

--info
Display certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

--tmpdir <tmp>, -t <tmp>
Specify the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>
Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

See Also

cov-analyze

cov-emit

cov-collect-models

Name

cov-manage-emit Manage an intermediate directory.

Synopsis

```
cov-manage-emit <GENERAL OPTIONS> <COMMANDS> <COMMAND OPTIONS>

GENERAL OPTIONS:

[ --cpp | --cs | --java]

{--dir <intermediate_directory>|--idir-library
<intermediate_directory_library>}

[--tu <tu_ids> | --tu-pattern <pattern>]

[--tus-per-psf <value>]

COMMANDS and COMMAND OPTIONS:

[add-other-hosts | check-integrity | delete-source | list-builds | repair |
reset-host-name

[add <int_dir> | list | list-json | preprocess | link-file <out_file>]

[{recompile | retranslate | retranslate-or-emit} <recompile_options>

[find [OPTIONS]

[add-coverage [OPTIONS] | compute-coverability { --verbose } | delete-
coverage | delete-test-coverage [OPTIONS] | [list-coverage-known | list-
coverage-unknown] --output <out_file> {--filename-regex <regex>} {--count} |
remove-coverability {--verbose} | list-tests {--suitename <name>} {--count}]

[list-compiled-classes]

[add-to-library --dir <intermediate_directory>]

[remove-from-library --dir <directory_name>]

[Shared options]
```

Description

The `cov-manage-emit` command is used to query and manipulate an emit repository. Each intermediate directory contains a single emit repository (created by `cov-build`, `cov-emit`, `cov-emit-cs`, or `cov-emit-java`) that contains data for C/C++, C# and Java compilations.

The `cov-manage-emit` command requires the `--dir` option, plus at least one sub-command. the `cov-manage-emit` command line typically follows this pattern:

```
cov-manage-emit <general_options> <sub-command> <sub-command_options>
```

The sub-commands can be used for various operations, including:

- Repairing database integrity (`repair`).
- Recompiling (`recompile`).
- Copying information from one intermediate directory into another (`add`).
- Aggregating the results of a distributed build into a single intermediate directory (`add-other-hosts`).
- Listing source files (`print-source-files`).
- Listing AST definitions (`find --print-definitions`).
- Adding test coverage data for Test Advisor (`add-coverage`).
- Inputing and outputting SCM data for Test Advisor (`add-scm-annotations`, `dump-scm-annotations`).
- Starting, stopping, and querying the emit server.
- Export C/C++ coverage data suitable for use with Test Advisor QA Edition.

The `cov-manage-emit` options are grouped by basic, options that cannot be filtered by translation units, options that require translation unit filtering, options for listing emit database information, and options for recompiling.

The `cov-manage-emit` command returns the following success or failure values:

- 0 – Success
- 2 – Error
- 4+ – Internal error, contact support@coverity.com

Options

Basic options

Either the `--dir` option or `--idir-library` option is required. If you use a sub-command that uses translation units, you can filter this information with the `--tu` or `--tu-pattern` option, or both.

--dir `<intermediate_directory>`
: Specifies an existing intermediate directory that was created with the `cov-build` command. While certain other sub-commands (for example, `add`) allow you to specify intermediate directories, the one specified with `--dir` is the directory modified by `cov-manage-emit`.

--idir-library <intermediate_directory_library>
Specifies the location of an intermediate directory library, which may contain multiple intermediate directories created with the `cov-build` command. Certain sub-commands, such as `start-server`, will optionally accept an intermediate directory library instead of a single intermediate directory.

--cpp
Filters by C/C++ translation units on which this command operates or reports. The command will fail with an informative error message if there are no C/C++ translation units in the emit.

--case-normalized-filename
By default, `cov-manage-emit` displays case-preserved file names in the output. Specifying this option allows `cov-manage-emit` to display normalized file names (that is, names that are entirely lower case).

For example (assuming you are on Windows and have a file in your emit named `MyFile.c`):

```
cov-manage-emit.exe --dir intdir --case-normalized-filename list
```

The output will include the following:

```
c:/cygwin/space/int_dir/myfile.c
```

Note

In previous releases, case-preserved file names were always printed for Java and C# (regardless of `--case-preserved-filename`. As of the 7.5.0 release, Java and C# file names will be case-preserved or case-normalized according to specification of this option (`--case-normalized-filename`), like C/C++ file names. As a result, it is impossible to get the old output of `cov-manage-emit` (which would case-normalize C/C++ but case-preserve Java/C#) in a multi-language scenario.

--case-preserved-filename
Allows `cov-manage-emit` to display case-preserved file names. This option is enabled by default, so you do not need to specify it with `cov-manage-emit list`. To switch to normalized file names, use `--case-normalized-filename`.

--cs
Filters by C# translation units on which this command operates or reports. The command will fail with an informative error message if there are no C# translation units in the emit.

--java
Filters by Java translation units on which this command operates or reports. The command will fail with an informative error message if there are no Java translation units in the emit.

--tu <translation_unit_id(s)>, -tu <translation_unit_id(s)>
Identifies a set of translation units (TUs), named by their numeric ID attribute(s). A translation unit approximately maps to the output from a single run of a compiler. This option requires a comma-separated list of id(s), and `--tu` can be specified multiple times. The union of all these identifier sets

is the set of TUs to operate on subsequently, for operations that work on TUs. It is an error if any of the specified IDs do not correspond to any existing translation unit. To get the IDs for translation units, use the list sub-command.

You can use the `--tu` and `--tu-pattern` options together.

--tu-pattern `<translation_unit_pattern>`, -tp `<translation_unit_pattern>`
Identifies a set of translation units specified with a translation unit pattern. The `--tu-pattern` option can be specified multiple times. Matching TU sets are unioned together across all patterns.

Both `--tu` and `--tu-pattern` can be specified on a single command line. The final set of TUs operated upon includes a given TU if it matches any specified translation unit pattern or its ID is listed explicitly as an argument to `--tu`.

It is an error if at least one `--tu-pattern` argument is specified but no translation unit matches any of the specified patterns.

You can get useful information on TUs with the list sub-command.

For more information, see Translation unit pattern matching.

--tus-per-psf `<value>`
Indicates how the set of primary source files affects the set of selected TUs. The possible values are as follows:

- `all`: Select all TUs, possibly as specified by other TU filters. This is the default.
- `latest`: Select only the latest TU with a given primary source file according to the time of compilation. If there are multiple TUs with the same primary source file within a single build, a deterministic TU is chosen within that build, regardless of time of compilation, which allows determinism in the case of parallel builds. This corresponds to the default set of TUs that `cov-analyze` analyzes. That is, `cov-analyze` with `--one-tu-per-psf` corresponds to `--tus-per-psf=latest` without any other filtering options (see the --one-tu-per-psf option to `cov-analyze`).

  With at least one `-tu` option and without a search pattern, the option has no effect. In this case, the system includes only the TUs specified with `-tu`.
- `non-latest`: Select any but the `latest` TU with a given primary source file. This is applied after search pattern filtering. The result is undefined if `-tu` is also used. For instance, to keep only one TU per primary source file, run the following command:

  ```
  cov-manage-emit --tus-per-sf=non-latest delete
  ```

Examples:

To list all the TUs that cov-analyze will operate on:

```
> cov-manage-emit --dir  <intermediate_directory> \
```

```
    --tus-per-psf=latest list
```

To delete TUs and leave only the ones that cov-analyze would operate on:

```
> cov-manage-emit --dir  <intermediate_directory> \
    --tus-per-psf=non-latest delete
```

Non-filtered sub-commands

These sub-commands cannot be filtered with translation unit options.

add-coverage <command_options>
Adds the coverage data contained in the specified file to the intermediate directory. Valid command options:

- `--batch <filename>`

  Parses the specified file as batch commands to run. If this command option appears on the command line, all other add-coverage command options on the command line are ignored.

  add-coverage batch file format:

  To efficiently add data from many separate coverage files, add-coverage supports batch files. Each line in a batch file specifies a single batch command to be executed by add-coverage, and batch commands are executed in the order of appearance in the batch file. The grammar for batch commands is:

  ```
  BatchCommand ::= 'run' | BatchOption
   BatchOption  ::= <option_name> ':' <length> ':' <value> <option_name>
  ```

  A BatchOption batch command sets the specified option to the specified value for all subsequent run batch commands. This value overrides any previous setting of the specified option. Valid option names are any command option available to the add-coverage command except for the `--batch` command option. If the command option accepts a value, the option is followed by a colon, then the string length of the value, then another colon, then the value itself. If the command option does not accept a value, only the option name is specified in the corresponding BatchOption. When used in a BatchOption batch command, the leading '`--`' is omitted from the option name.

  The run batch command causes a single coverage file to be read and its data added to the intermediate directory. The name of the coverage file to read from and other relevant settings are set by the BatchOption batch commands prior to the run batch command.

  The following is an example batch file:

  ```
  suitename:8:FooSuite
  testname:7:FooTest
  teststart:19:2012-03-19 07:12:11
  verbose
  gcda:9:test.gcda
  run
  gcda:10:test2.gcda
  ```

```
run
```

The above example batch file is equivalent to the following commands:

```
> cov-manage-emit --dir apache_2111 \
add-coverage --suitename FooSuite --testname FooTest \
--teststart "2012-03-26" --verbose --gcda test.gcda
```

```
> cov-manage-emit --dir apache_2111 \
 add-coverage --suitename FooSuite --testname FooTest \
--teststart "2014-03-26" --verbose --gcda test2.gcda
```

Note

Note that any test-specific options in the batch file will be replaced with their default values if they precede `testname` in the batch file. To avoid this, make sure the `testsource, teststart, teststatus`, and `testduration` options are specified *after* the `testname` option.

- `--bb <filename>`

  Reads the specified file as a file containing coverage data in `gcov bb` format. If this option is specified but the `--bbg` or `--da` option is not, the missing options will be inferred based on the filename specified in this option.

- `--bbg <filename>`

  Reads the specified file as a file containing coverage data in `gcov bbg` format. If this option is specified but the `--bb` or `--da` option is not, the missing options will be inferred based on the filename specified in this option.

- `--bullseye-csv`

  Specify the Bullseye CSV file from which to add coverage. This file can be generated from a Bullseye `.cov` file using the Bullseye `covbr` tool. This can be generated with the command:

  ```
  covbr --no-banner --quiet --csv --file <file.cov> --output <output.csv>
  ```

- `--bullseye-verbose`

  Enables verbose diagnostic messages in the Bullseye CSV parser.

- `--compilation-directory <dirname>`

  Uses the specified directory as the compilation directory. This is used to determine absolute paths of files referenced in the coverage data.

- `--coverage-file=<filename>`

  Where <filename> is the name of data captured using Coverity Function Coverage Instrumentation. See the Coverity Test Advisor and Test Prioritization 8.0 User and Administrator Guide for more information.

- The `--coverage-file` option cannot be used in combination with the `--batch` option.
- The `--testname, --suitename, --testsource, --testsource-encoding, --teststatus, --teststart`, and `--testduration` options are ignored when used with the `--coverage-file` option.

- `--coverage-selection <coverage-selection>`

  Describes what coverage is to be selected when merging coverage. For example:

  ```
  --coverage-selection "latest from all"
  ```

  see *Coverity Test Advisor and Test Prioritization 8.0 User and Administrator Guide*.

- `--da <filename>`

  Reads the specified file as a file containing coverage data in `gcov bbg` format. If this option is specified but the `--bb` or `--bbg` option is not, the missing options will be inferred based on the filename specified in this option.

- `--dry-run-list-tests`

  Indicates that execution will not actually merge coverage, but instead lists the tests which would be included in the target intermediate directory. This can be used to validate that a given coverage-selection option performs as intended.

- `--from-dir` <source-idir>

  This option indicates that `add-coverage` should use the specified intermediate directory as the source of coverage data to be merged. It is an error to use this option with `--from-idir-library`.

- `--from-idir-library` <mdir>

  This option indicates that `add-coverage` should use the intermediate directory library in `<mdir>` as the source of coverage data to be merged. It is an error to use this option with `--from-dir`.

- `--gcov <filename>`

  Reads the specified file as a file containing coverage data in `gcov` text format.

- `--gcov-version <version>`

  Parses `gcno`/`gcda` files that are derived from the specified version of gcc. This can be used to override the version declared in the file header in case this is incorrect. Valid values for `<version>` are of the form `x.y`, where `x` is the gcc major version number and `y` is the gcc minor version number. The patch level is ignored. For example, `gcc-4.6.3` would be specified with a version of `4.6`.

- `--gcov-verbose`

Enables verbose diagnostic messages in the `gcov` binary parser.

- `--gcno <filename>`

  Reads the specified file as a file containing coverage data in `gcov gcno` format. If this option is specified but the `--gcda` option is not, that option will be inferred based on the filename specified in this option.

- `--gcda <filename>`

  Reads the specified file as a file containing coverage data in `gcov gcda` format. If this option is specified but the `--gcno` option is not, that option will be inferred based on the filename specified in this option.

- `--purecov-text <filename>`

  Reads the specified file as a file containing coverage data in PureCoverage text format.

  By default, PureCoverage produces binary coverage files, however the tool has options to create text files instead, for example:

  For Linux:

  ```
  purecov -export=results.txt foo.pcv
  ```

  This command occurs after the instrumented binary execution as a post processing step.

  For Windows:

  ```
  purecov /SaveTextData=results.txt <executable>
  ```

  This occurs during the execution of the instrumented binary.

  For official instructions about how to produce a text file, see the PureCoverage documentation.

  The `compute-coverability` command option must be run after the build and before the analysis. This can be placed on the command line before or after the `add-coverage` subcommand.

- `--purecov-verbose`

  Enables verbose diagnostic messages in the Purecov parser.

- `--strip-path` <strip-path>

  Specifies an additional strip-path to use when merging coverage. This option can be specified multiple times.

  see *Coverity Test Advisor and Test Prioritization 8.0 User and Administrator Guide*.

- `--suitename <suitename>`

Identifies the coverage data as belonging to the named suite.

- `--testduration`

  The duration (in ms) of the test.

- `--testname <testname>`

  Identifies the coverage data as belonging to the named test.

- `--testsource`

  The path to the 'sourcefile' for the test. This could be a test script, a Makefile, or a source file of a unit test.

  If the argument ends with a colon (:) followed by one or more digits, then the digits shall be taken as the test source line number for the test, and only the portion of the argument before the colon shall be used for the test source filename. A default line number of 1 is used if no line number is provided.

- `--testsource-encoding`

  The encoding of the file provided to `--testsource`.

- `--teststatus [pass | fail | unknown]`

  Identifies the status of the test identified by `suitename` and `testname`. Must be one of pass, fail or unknown.

- `--teststart <teststart>`

  Identifies start time of the test identified by `suitename` and `testname`.

  See Accepted date/time formats for proper formatting of the `<teststart>` argument.

- `--windriver-run`

  Reads the specified file as a file containing coverage data in Wind River Coverage Run format. For implementation details, see Wind River VxWorks with Bullseye.

- `--verbose`

  Enables verbose diagnostic messages.

add-other-hosts
: Adds all translation units from emit repositories in the current intermediate directory but associated with host names other than the current one. In general, an intermediate directory can contain several emits, each associated with a specific host name. This option copies all of the TUs from emits associated with other hosts into the emit associated with the current host. This sub-command can be used to aggregate the results of a distributed build into a single intermediate directory.

add-scm-annotations --input <input_file>
Adds the SCM annotations for the source files in the specified input file to the source files in the intermediate directory. The input file can be the output file of the `cov-manage-emit dump-scm-annotations` option or the cov-extract-scm.

This option reads input from standard input when <input_file> is "-". Otherwise, it reads from the specified file.

Note

If you pipe the output of `cov-extract-scm`directly to `cov-manage-emit`, for example:

```
cov-extract-scm --input input.txt --output - |
cov-manage-emit --dir idir add-scm-annotations --input -
```

This will always generate at least one error and the first line of output will read `Extracting SCM data for ### files`.

add-test-capture-run [OPTIONS]
Adds a new test capture run to your intermediate directory. Note that in most use cases this is not required. The valid command options are as follows:

- `--build-id <build ID>`

  Specifies the build ID of the test capture run.

  For more information about `--build-id` usage, see the description for the cov-build --build-id option.

- `--set-as-current`

  Sets the test capture run as the "current" run in the intermediate directory. Any addition of coverage that does not specify a test capture run will use this test capture run.

- `--success <value>`

  Specify if this test capture run was successful. Valid values are:

  - `true`
  - `false`
  - `unknown`
  - `unset`

- `--test-capture-run-tag <tag>`

  Specifies a custom tag to allow for each selection of this test capture run. For example:

  ```
  "linux-build"
  ```

- `--test-capture-run-timestamp <timestamp>`

  Specifies the timestamp to use for the test capture run for this invocation. If it is not specified, the current time will be used, which is the typical use case. This option is only provided as a way for multiple test runs to use the same test capture run.

  See Accepted date/time formats for proper formatting of the `<timestamp>` argument.

add-to-library --dir `<intermediate_directory>`
: Adds an existing intermediate directory created by a `cov-build` command to the intermediate directory library specified with the `--idir-library` option. The intermediate directory library will be created if it does not exist.

  The added intermediate directory is uniquely identified within the library by directory name, which is the last non-empty component of the path to the intermediate directory.

  For example:

  ```
  > cov-manage-emit --idir-library libdir add-to-library --dir /home/user/my-idir
  ```

  The added intermediate directory is uniquely identified by "my-idir". This directory can then be referenced within the library using "my-idir".

check-compatible
: Checks if an emit version is compatible with the current Coverity Analysis tools. For example:

  ```
  cov-manage-emit --dir idir check-compatible
  ```

  This option returns 0 if it is compatible, 1 if it is not compatible, and 2+ if there is an error.

check-integrity
: Checks database integrity. If this check fails, print errors to stdout and exits with a non-zero code. Otherwise, exits with 0.

compute-coverability <command_options>
: Computes coverability of lines in files that are missing coverage data in the intermediate directory. The computed coverability is stored in the intermediate directory. Valid command options:

  - `--trace-log <filename>`

    Writes trace log messages to the specified file. Given a filename of `-` (a hyphen), the messages will be written to `stderr`. Any other value is interpreted as a file to open and write the log messages to.

  - `--verbose`

    Enables verbose diagnostic messages.

delete-bytecode
: Removes all file contents from the database. This is useful if you need to provide an intermediate directory to Coverity technical support and want to make sure that source code is excluded.

delete-coverage
: Removes all coverage data from the database. This is useful if you need to provide an intermediate directory to Coverity technical support and want to make sure that coverage data is excluded.

  To delete an individual test, see delete-test-coverage.

delete-scm-annotations
: Removes all SCM annotations from the database. This is useful if you need to provide an intermediate directory to Coverity technical support and want to make sure that SCM data is excluded.

delete-source
: Removes only source contents, including all webapp archive files (JSP, XML, and so forth). This option does not remove Java class, JAR files, or C# bytecode. This is useful if you need to provide an intermediate directory to Coverity technical support and want to make sure that the source content is excluded.

delete-test-coverage [OPTIONS]
: Deletes coverage for a specific test across all TUs affected by the test. The test to be deleted is uniquely identified by the following command options:

  - `--suitename <suitename>`

    Specifies the suite name belonging to the test. Suite names are available through the list-tests sub-command.

  - `--testname <testname>`

    Specifies the test name belonging to the test. Test names are available through the `list-tests` sub-command.

  - The test capture run, which you can specify with the `-test-capture-run-id` command option or by the combination of the following command options:

    - `--build-id <build ID>`

      Specifies the build ID of the test capture run.

      For more information about `--build-id` usage, see the description for the cov-build --build-id option.

    - `--test-capture-run-tag <tag>`

      Specifies the tag used to identify the test capture run.

    - `test-capture-run-timestamp <timestamp>`

      Specifies the time when the test capture run occurred.

    Information about test capture runs is available through the list-test-capture-runs sub-command. For more details on these options, including valid `<timestamp>` formats, see the update-test-capture-run sub-command.

Examples:

```
> cov-manage-emit --dir idir delete-test-coverage --suitename SuiteAlpha \
 --testname TestAlpha --test-capture-run-id 5
```

```
> cov-manage-emit --dir idir delete-test-coverage --suitename SuiteBeta \
 --testname TestBeta --test-capture-run-timestamp "2010-01-04T13:53:08" \
 --build-id 100 --test-capture-run-tag "CaptureTag"
```

To delete coverage for all tests, delete-coverage.

dump-scm-annotations --output <output_file>
Output all SCM annotations stored in the intermediate directory. This command creates a cache of the SCM annotations that is intended to be reapplied in the future using add-scm-annotations.

This option places output on a standard output when <output_file> is "-". Otherwise, it outputs to the specified file.

export-ta-qae-data <options>
Generate a scan (`.kss`) and/or footprint (`.ksa`) file for use with Test Advisor QA Edition. This sub-command is controlled by the following options:

- `--ksa-file <filename>`

  This is the name of the footprint (`.ksa`) file to generate. You must specify at least one of `--ksa-file` or `--kss-file`.

- `--kss-file <filename>`

  This is the name of the scan (`.kss`) file to generate. You must specify at least one of `--ksa-file` or `--kss-file`.

- `--release-name <string>`

  This is the name of the release that you are creating the scan file for. This should match what you enter in the cockpit when uploading the scan. For example: `1.0.`

- `--strip-path <strip-path>`

  The strip-path to use when evaluating filenames.

list-builds
Reports total number of successful and failed builds.

list-compiled-classes
Lists the classes contained in the emit that have been compiled. Use with `--java` or `--cs` to limit the results to one of the languages. The output is CSV-formatted and written to standard output and is designed to be specified into the `cov-build --java-instrument-classes <filename>` command.

The CSV file format has two columns:

- Column 1 - The name of the class.
- Column 2 - The full path to the source file for the class.

This sub-command applies only to Java and C#, not to C/C++.

list-coverage-known <command_options>
Lists the files contained in the emit which have corresponding coverage data included in the emit. The valid command options are:

- `--output <output_file>`

  The list is written to standard output when `<output_file>` is `"-"`. Otherwise, the list is written to the specified file.

- `--filename-regex <regex>`

  Includes a file for consideration if the regular expression (regex) matches the name of the file; this is not case-sensitive. For the purpose of turning a file name into a string that can then be matched against a regex, the following normalizations are applied:

  - The name is made absolute, including the drive letter on Windows systems.
  - The forward-slash character ("/") separates name components.
  - When no drive letter is present, the name begins with a forward-slash character ("/"); otherwise, a forward-slash character ("/") follows the drive letter.

- `--count`

  Reports the number of files that would have been reported. If used with `--filename-regex`, it reports the number of matching files only.

list-coverage-unknown <command_options>
Lists the source files contained in the intermediate directory which do not have corresponding coverage data included in the intermediate directory. The valid command options are:

- `--output <output_file>`

  The list is written to standard output when `<output_file>` is the dash character ("-"). Otherwise, the list is written to the specified file.

- `--filename-regex <regex>`

  Includes a file for consideration if the regular expression (regex) matches the name of the file; this is not case-sensitive.

  For the purpose of turning a file name into a string that can then be matched against a regex, the following normalizations are applied:

  - The name is made absolute, including the drive letter on Windows systems.

- The forward-slash character ("/") separates name components.
- When no drive letter is present, the name begins with a forward-slash character ("/"); otherwise, a forward-slash character ("/") follows the drive letter.

- `--count`

  Reports the number of files that would have been reported. If used with `--filename-regex`, it reports the number of matching files only.

list-functions-v1
Lists the functions in the intermediate directory. The valid command options are:

- `--function-pattern <pattern>`

  Restrict output to only those functions which match `<pattern>`. `<pattern>` follows the syntax described in the Translation unit pattern matching section, however the following predicates are used instead of the predicates described in that section:

  - `mangled_name(<regex>)`: The function is included if its mangled name matches the given regex.
  - `unmangled_name(<regex>)`: The function is included if its unmangled name matches the given regex.
  - `filename(<regex>)`: The function is included if it is in a file whose stripped name matches the given regex.
  - `impacted_since(<date>)`: The function is included if it was impacted on or after the given date. [FM]
  - `impacted()`: Like `impacted_since(<date>)`, but uses the `code-version-date` in the emit as the given date. [FM]
  - `directly_impacted_since(<date>)`: The function is included if it was directly impacted on or after the given date. [FM]
  - `directly_impacted()`: Like `directly_impacted_since(<date>)`, but uses the `code-version-date` in the emit as the given date. [FM]
  - `indirectly_impacted_since(<date>)`: The function is included if it was indirectly impacted on or after the given date. Functions without indirect impact dates are not included. [FM]
  - `indirectly_impacted()`: Like `indirectly_impacted_since(<date>)`, but uses the `code-version-date` in the emit as the given date. [FM]
  - `scm_modified_since(<date>)`: The function is included if it was modified on or after the given date. SCM data is used to determine modification. Functions without SCM data are included. [FM]

- `has_scm_data()`: The function is included if it has SCM data. [FM]
- `covered_by_suitename(<regex>)`: The function is included if it was executed by a test whose suitename matches the given regex. [TM]
- `covered_by_testname(<regex>)`: The function is included if it was executed by a test whose testname matches the given regex. [TM]
- `covered_by_test(<suite_regex>, <test_regex>)`: The function is included if it was executed by a test whose suitename matches `<suite_regex>` and whose testname matches `<test_regex>`. [TM]

Predicates marked with [FM] use function metrics generated by Test Advisor analysis, and require that such analysis be run prior to querying. Typically this is done by running `cov-analyze` with either the `--test-advisor` or `--enable-test-metrics` option.

Predicates marked with [TM] use test metrics information, and require that such analysis be run prior to querying. Typically this is done by running `cov-analyze` with the `--enable-test-metrics` option.

Function metrics and test metrics which have been generated in an intermediate directory using `cov-analyze` can be re-used within that directory for the purpose of this query. The `import` command of `cov-manage-history` should be run on the intermediate directory to allow this reuse.

- `--output-fields <fields>`

  Specifies the fields for each function to include in the output. `<fields>` is a comma-separated list of keywords from among the following:

  - `mangled_name`: The mangled name of the function.
  - `unmangled_name`: The unmangled name of the function.
  - `filename`: The name of the file containing the function.
  - `impact_date`: The impact date of the function. [FM]
  - `direct_impact_date`: The direct impact date of the function. [FM]
  - `indirect_impact_date`: The indirect impact date of the function. [FM]
  - `scm_modified_date`: The SCM modified date of the function. [FM]
  - `covering_tests`: The tests that cover the function. Requires the `--json` option be specified, see below. [TM]
  - `default`: The default output fields. This is equivalent to "`mangled_name,unmangled_name,filename`", and is the default if the `--output-fields` option is not specified.

[FM] and [TM] indicate output fields which use function metrics and test metrics, respectively, from the results of Test Advisor analysis. See `--function-pattern` above for details.

Output is in CSV format by default, unless the `--json` option is given. Each line except the first corresponds to a function, and contains the output fields in the order specified. The first line is a header indicating the field names.

- `--json`

  Output in JSON format. The output is a JSON array, where each element corresponds to a function. Each element of this array is an object whose name/value pairs correspond to the specified `--output-fields`.

  This option is required when `covering_tests` is included in the `--output-fields` option. Specifying the `covering_tests` output field will add an element named `covering_tests` to each function object, whose value is an array representing the tests that cover the function. Each element of this array is an object with the following name/value pairs:

  - `suitename`: The suitename of the test.
  - `testname`: The testname of the test.

- `--strip-path <path>`

  The strip-path to use when evaluating filenames. Normally this is not required, but can be used to override the default.

All dates must match a format described in Accepted date/time formats.

list-scm-known <command_options>
: Lists the files contained in the emit that have corresponding SCM data included in the emit. The valid command options are:

  - `--output <output_file>`

    The list is written to standard output when `<output_file>` is the dash character ("-"). Otherwise, the list is written to the specified file.

  - `--filename-regex <regex>`

    Includes a file for consideration if the regular expression (regex) matches the name of the file; this is not case-sensitive.

    For the purpose of turning a file name into a string that can then be matched against a regex, the following normalizations are applied:

    - The name is made absolute, including the drive letter on Windows systems.
    - The forward-slash character ("/") separates name components.

- When no drive letter is present, the name begins with a forward-slash character ("/"); otherwise, a forward-slash character ("/") follows the drive letter.

- `--count`

  Reports the number of files that would have been reported. If used with `--filename-regex`, it reports the number of matching files only.

list-scm-unknown <command_options>
Lists the source files contained in the intermediate directory that do not have corresponding SCM annotations included in the intermediate directory. The valid command options are:

- `--output <output_file>`

  The list is written to standard output when `<output_file>` is the dash character ("-"). Otherwise, the list is written to the specified file.

- `--filename-regex <regex>`

  Includes a file for consideration if the regular expression (regex) matches the name of the file; this is not case-sensitive.

  For the purpose of turning a file name into a string that can then be matched against a regex, the following normalizations are applied:

  - The name is made absolute, including the drive letter on Windows systems.
  - The forward-slash character ("/") separates name components.
  - When no drive letter is present, the name begins with a forward-slash character ("/"); otherwise, a forward-slash character ("/") follows the drive letter.

- `--count`

  Reports the number of files that would have been reported. If used with `--filename-regex`, it reports the number of matching files only.

list-test-capture-runs
Lists the test capture runs currently stored in the intermediate directory or intermediate directory library.

list-tests <command_options>
Lists all tests which have been stored in the intermediate directory. The valid command options are:

- `--count`

  Reports the number of tests that would have been reported. If used with `--suitename`, it reports the number of tests for suite only.

- `--suitename <name>`

Restricts the tests that are listed or counted to those that belong to the given named suite.

list-tests-v2

Lists the tests in the intermediate directory. The valid command options are:

- `--test-pattern <pattern>`

  Restrict output to only those tests which match `<pattern>`. `<pattern>` follows the syntax described in the Translation unit pattern matching section, however the following predicates are used instead of the predicates described in that section:

  - `suitename(<regex>)`: The test is included if its `suitename` matches the given regex.
  - `testname(<regex>)`: The test is included if its `testname` matches the given regex.
  - `covers_function_mangled_name(<regex>)`: The test is included if it covers at least one line of a function whose mangled name matches the given regex. [TM]
  - `covers_function_unmangled_name(<regex>)`: The test is included if it covers at least one line of a function whose unmangled name matches the given regex. [TM]
  - `covers_filename(<regex>)`: The test is included if it covers at least one line of a file whose stripped filename matches the given regex. [TM]

  Predicates marked with [TM] use test metrics information, and require that such analysis be run prior to querying. Typically this is done by running `cov-analyze` with the `--enable-test-metrics` option.

  Test metrics which have been generated in an intermediate directory using `cov-analyze` can be re-used within that directory for the purpose of this query. The import command of `cov-manage-history` should be run on the intermediate directory to allow this reuse.

- `--output-fields <fields>`

  Specifies the fields for each test to include in the output. `<fields>` is a comma-separated list of keywords from among the following:

  - `suitename`: The suitename of the test.
  - `testname`: The testname of the test.
  - `run_date`: The date of the latest run of the test.
  - `status`: The status of the latest run of the test. This is a string from the set {"pass", "fail", "unknown"}.
  - `duration_ms`: The (integer) duration of the latest run of the test in milliseconds.
  - `source_filename`: The stripped filename of the source of the test.

- `source_line`: The 1-based line number of the source of the test.
- `test_capture_run_id`: The (integer) id of the `TestCaptureRun` for the test.
- `covered_functions`: The tests that cover the function. Requires the `--json` option be specified, see below. [TM]
- `default`: The default output fields. See below.

[TM] indicates output fields which use test metrics from the results of Test Advisor analysis. See `--test-pattern` above for details.

Output is in CSV format by default, unless the `--json` option is given. Each line except the first corresponds to a test, and contains the output fields in the order specified. The first line is a header indicating the field names.

- `--json`

  Output in JSON format. The output is a JSON array, where each element corresponds to a test. Each element of this array is an object whose name/value pairs correspond to the specified `--output-fields`.

  This option is required when `covered_functions` is included in the `--output-fields` option. Specifying the `covered_functions` output field will add an element named `covered_functions` to each test object, whose value is an array representing the functions covered by the test. Each element of this array is an object with the following name/value pairs:

  - `mangled_name`: The mangled function name.
  - `unmangled_name`: The unmangled function name.
  - `filename`: The name of the file containing the function.

- `--strip-path <path>`

  The strip-path to use when evaluating filenames. Normally this is not required, but can be used to override the default.

All dates must match a format described in Accepted date/time formats.

query-build-id
: Outputs the current build ID for this intermediate directory.

remove-coverability <command_options>
: This option is deprecated as of the 8.0 release. Use `delete-coverage` instead.

  Removes computed coverability of lines in files in the intermediate directory. This is effectively the inverse of compute-coverability. Valid command options:

  - `--verbose`

    Enables verbose diagnostic messages.

remove-from-library --dir `<directory_name>`
: Removes an intermediate directory from the intermediate directory library specified with the `--idir-library` option.

  For example, the following command will remove the intermediate directory "my-idir" from the library "libdir".

```
> cov-manage-emit --idir-library libdir remove-from-library --dir my-idir
```

repair
: Repairs database integrity. This operation might cause data loss, such as discarding translation units that are damaged.

reset-host-name
: If the specified intermediate directory has data associated with a single host name other than the current host name, changes the host name associated with the emit database to the current host name.

update-test-capture-run
: Updates an existing test capture run in your intermediate directory. The valid command options are:

  - `--build-id <build ID>`

    Specifies the build ID of the test capture run.

    For more information about `--build-id` usage, see the description for the cov-build --build-id option.

  - `--success <value>`

    Specify if this test capture run was successful. Valid values are:

    - `true`
    - `false`
    - `unknown`
    - `unset`

  - `--set-as-current`

    Sets the test capture run as the "current" run in the intermediate directory. Any addition of coverage that does not specify a test capture run will use this test capture run.

  - `--test-capture-run-id <TCR ID>`

    Specifies the test capture run ID of the test capture run to use. You can determine the test capture run ID for a specific run by using the list-test-capture-runs sub-command.

  - `--test-capture-run-tag <tag>`

Specifies a custom tag to allow for each selection of this test capture run. For example:

```
"linux-build"
```

- `--test-capture-run-timestamp <timestamp>`

  Specifies the timestamp to use for the test capture run for this invocation. If it is not specified, the current time will be used, which is the typical use case. This option is only provided as a way for multiple test runs to use the same test capture run.

  See Accepted date/time formats for proper formatting of the `<timestamp>` argument.

Translation unit sub-commands with optional filtering

The following filtering sub-commands work on translation units. By default, all translation units are included in the results. You can optionally restrict the translation units used in these operations with the `--tu` and/or `--tu-pattern` options.

The options for listing emit database information and recompiling also support restricting the translation units with the `--tu` and/or `--tu-pattern` options.

add `<int_dir>`
: Add (copy) all translation units from a specified intermediate directory (`<int_dir>`) into the current one (the one specified with the `--dir` option). If `--tu` and/or `--tu-pattern` are specified, then those filters are interpreted as applying to the source emit, and only the matching subset is copied.

link-file `<out_file>`
: Create a file (`<out_file>`) with a description of the specified translation units as a link file, which can be used as input to `cov-link`.

list
: List all translation units in the intermediate directory. Each translation unit is identified by its numeric ID, which is listed along with its primary source file name.

list-json
: List all translation units in the intermediate directory as a standards-compliant JSON array. The translation units are identified by a numeric ID, which is listed along with the following fields:

  - `id:` The unique numeric translation unit ID.
  - `primaryFilename:` The primary source file name.
  - `primaryFileSizeInBytes:` The size of the primary source file in bytes.
  - `primaryFileHash:` MD5 hash of the contents of the primary source file.
  - `language:` String describing the translation unit language.
  - `userLanguage:` The user-specified translation unit language.

- `hasASTs:` Boolean. 'true' if the intermediate directory contains an AST for this translation unit, otherwise 'false'.
- `mspchTuFilename (optional):` The Microsoft precompiled header file which was created when this translation unit was built. This field is only displayed when a Microsoft precompiled header was created.
- `mspchId (optional):` The translation unit ID for the included Microsoft precompiled header. This field is only displayed when a precompiled header was used.

Example output:

```
[
  {
    "id" : 1,
    "primaryFilename" : "/home/build/project/tu1.cpp”,
    "primaryFileSizeInBytes" : 92,
    "primaryFileHash" : "6f700a28a47e79cddff8fba60cac7098",
    "language" : "C++”,
    "userLanguage" : "C++”,
    "hasASTs" : true
  },
  {
    "id" : 2,
    "primaryFilename" : "c:/project/stdafx.cpp",
    "primaryFileSizeInBytes" : 122,
    "primaryFileHash" : "3827e3e7426ce0bdebb7e51c94d2a680",
    "language" : "C++",
    "userLanguage" : "C++",
    "hasASTs" : false,
    "mspchTuFilename" : "c:/project/stdafx.pch"
  }
]
```

Note

The output of this command may contain additional attributes that are not documented here. For maximum interoperability, please ignore any attribute that is not documented.

Translation unit sub-commands with required filtering

The following filtering sub-commands work on translation units. You must supply the translation units used in these operations with the `--tu` and/or `--tu-pattern` options. The TU `list` sub-command identifies the TUs available for your required filter.

delete
: Delete all TUs that satisfy the specified translation unit filter.

preprocess
: Similar to `recompile`, except that when `cov-emit` is invoked, it is passed the `-E` (preprocess) and `--output_defs` options, which results in preprocessing only. The emit database is not modified by this operation.

The preprocessed output file (which is the stdout of `cov-emit`) is stored in the `preprocessed` subdirectory of the `c/output` subdirectory of the intermediate directory. The name of the file is the name of the primary source file for the TU, minus any path information, minus any file extension, plus either `.i` or `.ii`.

This option works only for C/C++ source code, not Java.

print-compilation-info [<`options`>]
: For the specified translation units, print the command lines for `cov-emit`, `cov-translate` (if it was run), and `cov-build` (if it was run).

  The options are:

  - `--detailed` - provides all process details except environment variables.
  - `--print-env` - provides the environment variable definitions for the process.

print-compilation-time
: Prints the invocation time of any `cov-emit`, `cov-emit-java`, `cov-emit-cs`, `cov-translate`, or `cov-build` for a given translation unit (TU) to be easily accessible.

  Usage examples

```
cov-manage-emit --dir dir -tu <TU#> print-compilation-time
```

```
cov-manage-emit --dir dir -tp <pattern> print-compilation-time
```

  Output example:

```
cov-manage-emit --dir idir -tp "success()" print-compilation-time

Looking for translation units
|0----------25----------50----------75---------100|
**************************************************
Translation unit:
1 -> /Users/emoriarty/Testing/BZ55606/test.cpp
cov-emit invocation time (seconds): 1
cov-translate invocation time (seconds): 1
cov-build invocation time (seconds): 2
Translation unit:
2 -> /Users/emoriarty/Testing/BZ55606/test.cpp
cov-emit invocation time (seconds): 2
cov-translate invocation time (seconds): 2
cov-build invocation time (seconds): 2
Translation unit:
3 -> /Users/emoriarty/Testing/BZ55606/test.cpp
cov-emit invocation time (seconds): 2
cov-translate invocation time (seconds): 2
cov-build invocation time (seconds): 2
Translation unit:
4 -> /Users/emoriarty/Testing/BZ55606/test.cpp
cov-emit invocation time (seconds): 2
cov-translate invocation time (seconds): 2
```

```
cov-build invocation time (seconds): 2
```

print-source
: For the specified translation units, list the name and the contents of the primary source file associated with the TU. This option also reports, in parentheses, the internal row ID of the source file. It accepts the same command options as `print-source-files-contents`.

print-source-files
: For the specified translation units, list the names of all the source files associated with the TU. Also reports, in parentheses, the internal row ID of the source file.

print-source-files-contents
: For the specified translation units, list the names and contents of all the source files associated with the TU. Also reports, in parentheses, the internal row ID of the source file.

  `print-source-files-contents` has the following options:

  - `--scm-annotations` - Prefixes each source line with the change record (or commit record) that contributed most recently to the line. The change record data that is as follows:

    Date and time that the change record was applied according to the SCM system.

    The author (username) attributed to the change record.

    The revision of the change, which is an identifier for the change record provided by the SCM system.

  - `--coverage` - Outputs and prefixes each line of the TU being changed. Each line is printed with a marker identifying whether or not that line has been covered by a test (as seen by the Test Advisor `cov-capture` command). The notations is as follows:

    `"+"` - The line has been covered by one or more tests.

    `"-"` - The line has not been covered by any tests.

    `" "` - The line is not eligible for coverage (that is, the line does not represent executable code).

    `"?"` - The line does not have coverage data available.

    For example:

```
Translation unit:
1 -> ./sample.cpp
  Primary SF : ./sample.cpp (row ID 1)
#/* Sample that generates interesting coverage lines */
#/* (c) 2015 Synopsys, Inc. All rights reserved worldwide. */
#
+#void call_seven_times()
#{
+#}
#
+#void call_three_times()
```

```
#{
+#}
#
+#void call_ten_times()
#{
+#}
#
-#void call_zero()
#{
-#}
#
+#int main()
#{
+#  int x = 0;
+#  for (int i = 0; i < 10; ++i) {
#    x += i
+#       * 3;  /* multi-line statement, may be wrong */
+#    if (i < 7) {
+#      call_seven_times();
#    } else {
+#      call_three_times();
#    }
+#    call_ten_times();
+#    if (x > 300) {
-#      call_zero();
#    }
#  }
+#  return 0;
#}
```

- `--build-id <build ID>`

  Specifies the build ID of the test capture run.

  For more information about `--build-id` usage, see the description for the cov-build --build-id option.

- `--test-capture-run-id <TCR ID>`

  Specifies the test capture run ID of the test capture run to use. You can determine the test capture run ID for a specific run by using the list-test-capture-runs sub-command.

- `--test-capture-run-tag <tag>`

  Specifies a custom tag to allow for each selection of this test capture run. For example:

  ```
  "linux-build"
  ```

- `--test-capture-run-timestamp <timestamp>`

  Specifies the timestamp to use for the test capture run for this invocation. If it is not specified, the current time will be used, which is the typical use case. This option is only provided as a way for multiple test runs to use the same test capture run.

See Accepted date/time formats for proper formatting of the `<timestamp>` argument.

- `--suitename <suitename>`

  Identifies the coverage data as belonging to the named suite. This option is an accepted option when `--coverage` is used.

- `--testname <testname>`

  Identifies the coverage data as belonging to the named test. This option is an accepted option when `--coverage` is used.

print-source-files-stats
: For the specified translation units, list the names of all of the source files associated with the TU. Also reports the internal row ID of the source file (in parentheses) followed by statistics for that source file. The statistics listed include:

  - The file contents time stamp, size, and MD5 sum
  - The count of blank lines
  - The count of comment lines
  - The count of code lines
  - The count of code lines with inline comments

  Example output is as follows:

```
1 -> /example_dir/a.cpp
  Primary SF : /example_dir/a.cpp (row ID 1)
    Timestamp:     2013-07-19 11:37:35
    Size:          25
    MD5 sum:       7edc175dc475923c51c579924b724a8c
    Blank lines:   1
    Comment lines: 0
    Code lines:    2 (1 with inline comments)
```

print-tuid
: Prints the TU ids for the TU requested using either `-tu` or `--tu-pattern`. Unlike most commands that will error if an invalid tu is specified, `print-tuid` will silently ignore it. For example:

```
$ cov-manage-emit --dir foo --tu-pattern 'success()' print-tuid
Looking for translation units
|0----------25----------50----------75---------100|
 ************************************************
 1
 3
 4
```

```
$ cov-manage-emit --dir foo --tu-pattern 'success()' print-tuid -of tuids.txt
Looking for translation units
|0----------25----------50----------75---------100|
 ************************************************
 $ cat tuids.txt
 1
 3
 4
```

Listing emit database information

The `find` sub-command lists information stored in the emit DB such as symbol names, locations, and definitions. By default, all translation units are included in the results. You can optionally restrict the translation units used in these operations with the `--tu` and/or `--tu-pattern` options.

What is being matched by the regular expression (regex) is, in C++, the mangled name of the symbol (according to the IA64 C++ ABI, see http://mentorembedded.github.io/cxx-abi/abi-examples.html#mangling), of which the actual identifier is always a substring. In C, what is matched is just the identifier.

find <`regular expression`> [OPTIONS]
: There are four kinds of symbols: functions, classes, global variables, and enumerations. If you specify

```
find <regular expression>
```

then the listing for the matching regex command is the symbol name, the kind of entity, the declaration location, and the definition TU.

You can control the information that is returned by using the following options:

--kind {f | c | e | g}
: Restricts the search to certain types of entities. The choices are `f`, `c`, `e` and `g`, for function, class, enum, and global, respectively.

--print-callees
: For a function, lists the set of functions it calls. Does not list information on other entities.

--print-definitions
: Lists the entity's definition syntax by pretty-printing the AST definition.

--print-debug
: Lists the entity's AST definition in debug (indented tree) mode.

The `find` sub-command accepts multiple operands and applies each of them as an inclusive filter when searching for symbols. In the following example, the first invocation of `cov-manage-emit` displays all symbols (global_1 and global_2). .

```
$ cat t.c
int global_1 = 1;
int global_2 = 2;
```

```
$ cov-emit --dir covint t.c
Emit for file '/tmp/t.c' complete.

$ cov-manage-emit --dir covint find .
Matching global: global_1
 declared at:
   /tmp/t.c:1:5-/tmp/t.c:1:12
 defined in TU 1 with row 1
Matching global: global_2
 declared at:
   /tmp/t.c:2:5-/tmp/t.c:2:12
 defined in TU 1 with row 2
```

The following two examples supply a regex command that selects exactly one of those symbols.

```
$ cov-manage-emit --dir covint find global_1
Matching global: global_1
 declared at:
   /tmp/t.c:1:5-/tmp/t.c:1:12
 defined in TU 1 with row 1

$ cov-manage-emit --dir covint find global_2
Matching global: global_2
 declared at:
   /tmp/t.c:2:5-/tmp/t.c:2:12
 defined in TU 1 with row 2
```

The following invocation specifies multiple regex commands that select both symbols.

```
$ cov-manage-emit --dir covint find global_1 global_2
Matching global: global_1
 declared at:
   /tmp/t.c:1:5-/tmp/t.c:1:12
 defined in TU 1 with row 1
Matching global: global_2
 declared at:
   /tmp/t.c:2:5-/tmp/t.c:2:12
 defined in TU 1 with row 2
```

Emit Server sub-commands

These commands allow for starting and stopping an emit server on the specified intermediate directory/ intermediate directory library. These commands all take an intermediate directory or an intermediate directory library.

You may only specify one of `--dir` or `--idir-library`, but not both.

start-server [OPTIONS]
: Starts an emit server for the specified intermediate directory or intermediate directory library. Valid options are:

  - `--port <port-number>`

Specifies the port number to bind the server. The default is 15772.

- `--interface <ip-address-to-run-on>`

  Specifies the IP address to which you want the server to bind.

- `--gcov-cache-size <size-in-mb>`

  Specifies the size of the cache in MB to use for gcov data. The default is 500MB.

- `--force-start`

  Forces the start of the server. This is useful if the previous emit server was not cleanly shut down and the PID file remains from the previous run.

Examples:

To start an emit server on a single intermediate directory:

```
cov-manage-emit --dir idir start-server [--port 15772] [--interface host_or_ip]
```

To start an emit server on an intermediate directory library:

```
cov-manage-emit --idir-library idir-lib start-server [--port 15772] [--interface
 host_or_ip]
```

stop-server

Stops a running emit server for the specified intermediate directory or intermediate directory library.

Examples:

To stop a running emit server on a single intermediate directory:

```
cov-manage-emit --dir idir stop-server
```

To stop an emit server on an intermediate directory library:

```
cov-manage-emit --idir-library idir-lib stop-server
```

query-server

Query an intermediate directory or intermediate directory library to verify that an emit server is already running. The output will be in a JSON format, for example:

```
[
  {
    "host" : "localhost",
    "pid" : 654321,
    "port" : 33445
  }
]
```

If no server is running, the JSON file will be empty.

Examples:

To check if an emit server is running for a given single intermediate directory:

```
cov-manage-emit --dir dir query-server
```

To check if an emit server is running for a given intermediate directory library:

```
cov-manage-emit --idir-library idir-lib query-server
```

Recompiling

The `recompile` sub-commands repeat a `cov-emit` compilation. You can use this option, for example, with updated `cov-emit` binary or compiler configuration settings to attempt to compile inputs that have previously failed. This is similar to `cov-build --replay`.

You can modify the translation units that are recompiled with the `--tu` and/or `--tu-pattern` options.

The `recompile` sub-commands are:

parse-source-only-tus [OPTIONS]
: Recompiles source-only TUs from the intermediate directory (those that were added through `cov-build --record-with-source` and that have not been recompiled already).

  This subcommand works only for C/C++ source code, not Java or C#.

recompile [OPTIONS]
: Recompile the set of TUs specified by the filter. For each TU to be recompiled, invoke `cov-emit` with the command line, environment settings, and current directory recorded in the emit repository. Source files are re-read from the file system.

recompile-from-dir [OPTIONS]
: Recompiles translation units from source contained within the emit directory. Replaying from the emit will have the same results, regardless of changes to the files in the filesystem (including deletion).

  This option is silimar to `cov-build --replay from emit`, but it allows you to perform finer-grained filtering of the TUs being replayed. For example:

  ```
  cov-manage-emit --dir idir --tu 10 recompile-from-dir
  ```

  This subcommand works only for C/C++ source code, not Java or C#.

replay-from-script -if <`json_file`> [OPTIONS]
: Reads a JSON script produced by Incredibuild, builds a list of compile commands, and executes each of the compile commands against `cov-translate` for accelerating Windows code builds using Incredibuild.

  The `-if <json_file>` option points to the json script file that is described in the `replay_from_script` command.

  For more information, see "Using IncrediBuild" in the Coverity Analysis 8.0 User and Administrator Guide. .

Note

`--record-only` works the same as "`cov-build --record-only`," recording the build to be replayed later.

retranslate [OPTIONS]
Run `cov-translate` on the set of TUs specified by the filter.

For each TU to be recompiled, invoke `cov-translate` using the command line, environment settings, and current directory recorded in the emit repository. Does not work with a TU complied directly by `cov-emit`.

Invocation of `cov-translate` requires a Coverity configuration. By default, the configuration that was used during the initial compilation will be used, but this can be overridden by specifying a configuration on the `cov-manage-emit` command line.

This subcommand works only for C/C++ source code, not Java or C#.

retranslate-or-emit [OPTIONS]
Run `cov-translate` on the set of TUs specified by the filter.

Similar to the `retranslate` option, except that in the case of a TU where `cov-emit` was invoked directly without `cov-translate`, invokes `cov-emit` instead of using `cov-translate`.

This subcommand works only for C/C++ source code, not Java or C#.

The `recompile` sub-command [OPTIONS] are as follows:

--compilation-log <log_file>
Saves diagnostic messages from `cov-translate` and `cov-emit` to <log_file> (instead of the default of standard output and standard error). Also displays a progress ticker bar.

--desktop
Used in conjunction with Desktop Analysis to perform recompilation faster by disabling bytecode decompilation in Java and C# builds.

--do-decomp
Used in conjunction with Desktop Analysis to perform recompilation in Java builds where bytecode decompilation is enabled.

--name <name>
Associates any new TUs created with a build named <name>. New TUs are not created by `parse-source-only-tus` or `recompile-from-dir`. These commands will reuse the existing TUs, so this option will have no effect. TUs will also not be created if the TUs are already up to date.

--parallel <number_of_processes> , -j <number_of_processes>
Spawn up to <number_of_processes> processes to run the recompilations. This option accepts the number of processes, or `auto` which sets the number of replay processes to the number of logical processors in the machine (`-j 0` is also accepted and is the same as `auto`).

Translation unit pattern matching

The argument to `--tu-pattern` is a string that acts as a filter on translation units. Alternatively, to use a file name for a pattern, specify `@<filename>`. Each pattern in this file must be on a separate line.

To get useful information about the translation units in an emit repository, use the list sub-command.

A pattern has the following syntax:

```
[!] <function>("<regex>"|'<regex>') [|| <function>("<regex>")]
    [&& <function>("<regex>")]
```

When combining patterns, the precedence from lowest to highest, is OR (`||`), AND (`&&`), and NEG (`!`). OR and AND are left-associative. You can use parentheses to group expressions to override precedence or associativity. The `regex` is a Perl regular expression. A backslash in a quoted string is interpreted as a regular expression metacharacter, and not as a string literal metacharacter. You can use single or double quotes to pass the string properly from the shell to the command. The Perl `regex` is used for partial matches; for full matches use the beginning of line (^) and end of line ($) symbols.

The values for `<function>` for where to apply the regular expression are:

arg
: Matches if the `regex` matches any of the native compiler command line elements, including the native compiler executable itself.

build_arg:
: Matches if the `regex` matches any argument to `cov-build`, including the `cov-build` executable name.

build_name("`<regex>`")
: Matches if `cov-build --name <name>` compiled the translation unit and `<name>` is matched by `<regex>`.

cov_emit_arg
: Matches if the `regex` matches any argument to the Coverity compiler front end, such as `cov-emit`, including the executable name.

file
: Matches if the `regex` matches the name of the primary source file. For the purpose of turning a file name into a string that can then be matched against a `regex`, the following normalizations are applied:
  - The name is converted to an absolute pathname. On Windows, this includes the drive letter.
  - On Windows, all letters are lower-cased, including the drive letter (this applies to all names in translation units created on Windows).
  - The forward-slash character (`/`) separates name components.
  - When no drive letter is present, the name begins with `/`; otherwise, a `/` follows the drive letter.

  For example:

  ```
  --tu-pattern "file('test\.c$')"
  ```

failure
: No argument. Matches if the compilation was unsuccessful (exit code != 0). Used by `cov-build --replay-failures`.

header
: Matches if the `regex` matches the name of any header file, which is defined to be a source file included in the TU other than the primary source file.

is_c("true|false")
: Matches if the translation unit was compiled as C source code and `is_c` is set to `true`, or, if the translation unit was not compiled as C source code and `is_c` is set to `false`.

is_csharp("true|false")
: Matches if the translation unit was compiled as C# source code and `is_csharp` is set to `true`, or, if the translation unit was not compiled as C# source code and `is_csharp` is set to `false`.

is_cxx("true|false")
: Matches if the translation unit was compiled as C++ and `is_cxx` is set to `true`, or, if the translation unit was not compiled as C++ and `is_cxx` is set to `false`.

is_java("true|false")
: Matches if the translation unit was compiled as Java source code and `is_java` is set to `true`, or, if the translation unit was not compiled as Java source code and `is_java` is set to `false`.

is_objc("true|false")
: Matches if the translation unit was compiled as Objective-C source code and `is_objc` is set to `true`, or, if the translation unit was not compiled as Objective-C source code and `is_objc` is set to `false`.

is_objcxx("true|false")
: Matches if the translation unit was compiled as Objective-C++ source code and `is_objcxx` is set to `true`, or, if the translation unit was not compiled as Objective-C++ source code and `is_objcxx` is set to `false`.

link_file
: The `regex` is not interpreted as a regular expression, but rather as the name of a file, which should be the output of `cov-link` (or `cov-manage-emit link_file`).

success
: No argument. Matches if the compilation was successful (exit code = 0).

Shared options

--debug, -g
: Turn on basic debugging output.

--info
: Display certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>
Set the detail level of command messages. Higher is more verbose (more messages). The default is 1. Use `--verbose 0` to disable progress bars.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

Examples

List build information from an intermediate directory:

```
> cov-manage-emit --dir apache_2111 \
    list-builds
```

List all translation unit information from an intermediate directory:

```
> cov-manage-emit --dir apache_2111 \
    list
```

List information from an intermediate directory for the translation unit with the ID 6:

```
> cov-manage-emit --dir apache_2111 --tu 6 \
    list
```

List all information on all entities:

```
> cov-manage-emit --dir apache_2111 \
    find  '.*'
```

List only callee information for all entities:

```
> cov-manage-emit --dir apache_2111 \
    find --print-callees  '.*'
```

List all information for entity `uninit`:

```
> cov-manage-emit --dir apache_2111 \
```

```
    find  '^uninit$'
```

List the definition of entity `uninit`:

```
> cov-manage-emit --dir apache_2111 \
    find  '^uninit$' --print-definitions
```

List the source files of TU 1:

```
> cov-manage-emit --dir apache_2111 \
    --tu 1 print-source-files
```

List TUs where `bar.cc` or `foo.cc` is the primary source file:

```
> cov-manage-emit --dir apache_2111 \
    --tu-pattern "file('bar.\cc$') || file('foo.\cc$')" list
```

List TUs using patterns specified in the file `files`:

```
> cov-manage-emit --dir apache_2111 \
    --tu-pattern @files list
```

Recompile the TUs in the emit database:

```
> cov-manage-emit --dir apache_2111  recompile
```

Recompile and put diagnostic information in the `211_log.txt` file:

```
> cov-manage-emit --dir apache_2111 \
    recompile --compilation-log 211_log.txt
```

Recompile only TU 1:

```
> cov-manage-emit --dir apache_2111 \
    --tu 1 recompile --compilation-log 211_log.txt
```

Recompile translation units where `test.c` is the primary source file:

```
> cov-manage-emit --dir apache_2111 \
        --tu-pattern "file('test\.c$')" recompile
```

List source files missing SCM annotations from the intermediate directory:

```
> cov-manage-emit --dir apache_2111 \
    list-scm-unknown --output files-without-scm-age-data.txt
```

List source files under `/builds` missing SCM annotations from the intermediate directory:

```
cov-manage-emit --dir apache_2111 list-scm-unknown  \
       --output files-without-scm-annotation-data.txt \
       --filename-regex '^/builds/'
```

List source files with existing SCM annotations in the intermediate directory:

```
> cov-manage-emit --dir apache_2111 \
```

```
    list-scm-known --output files-with-scm-age-data.txt
```

List source files under `/builds` with existing SCM annotations from the intermediate directory:

```
 cov-manage-emit --dir apache_2111 \
       list-scm-known --output files-with-scm-annotations.txt \
       --filename-regex '^/builds/'
```

Count the source files under `/builds` with existing SCM annotations from the intermediate directory:

```
cov-manage-emit --dir apache_2111 \
      list-scm-known --output count-files-with-scm-annotations.txt \
      --filename-regex '^/builds/' --count
```

Add SCM annotations for source files in the intermediate directory:

```
> cov-manage-emit --dir apache_2111 \
     add-scm-annotations --input scm-age-data.txt
```

Dump SCM annotations for source files in the intermediate directory:

```
> cov-manage-emit --dir apache_2111 \
    dump-scm-annotations --output scm-age-data.txt
```

Remove all SCM annotations from the intermediate directory:

```
cov-manage-emit --dir apache_2111 delete-scm-annotations
```

Add test coverage data from a single gcda file to the intermediate directory:

```
> cov-manage-emit --dir apache_2111 \
     add-coverage --suitename FooSuite --testname FooTest \
       --teststart "2012-03-19 07:12:11" --verbose \
       --gcda test.gcda
```

Add test coverage data in batch mode using the batch script coverage-batch.txt:

```
> cov-manage-emit --dir apache_2111 \
     add-coverage --batch coverage-batch.txt
```

Compute the missing coverage and add it to the intermediate directory:

```
> cov-manage-emit --dir apache_2111 compute-coverability
```

Remove all coverage from the intermediate directory:

```
cov-manage-emit --dir apache_2111 delete-coverage
```

Remove the coverage that was added by compute-coverability from the intermediate directory:

```
> cov-manage-emit --dir apache_2111 remove-coverability
```

List all tests that have results stored in the intermediate directory:

```
> cov-manage-emit --dir apache_2111 list-tests
```

List all tests for suite `MyModuleTests` that have results stored in the intermediate directory:

```
cov-manage-emit --dir apache_2111 list-tests --suitename MyModuleTests
```

Count all tests that have results stored in the intermediate directory:

```
cov-manage-emit --dir apache_2111 list-tests --count
```

List source files missing coverage data from the intermediate directory:

```
cov-manage-emit --dir apache_2111 \
       list-coverage-unknown --output files-without-coverage-data.txt
```

List source files under `/builds` missing coverage data from the intermediate directory:

```
cov-manage-emit --dir apache_2111 \
        list-coverage-unknown --output files-without-coverage-data.txt \
       --filename-regex '^/builds/'
```

List source files with existing coverage data in the intermediate directory:

```
cov-manage-emit --dir apache_2111 \
       list-coverage-known --output files-with-coverage-data.txt
```

List source files under `/builds` with existing coverage data from the intermediate directory:

```
cov-manage-emit --dir apache_2111 \
       list-coverage-known --output files-with-coverage-data.txt \
       --filename-regex '^/builds/'
```

Count the source files under `/builds` with existing coverage data from the intermediate directory:

```
cov-manage-emit --dir apache_2111 \
       list-coverage-known --output count-files-with-coverage-data.txt \
       --filename-regex '^/builds/' --count
```

See Also

cov-build

cov-extract-scm

cov-import-scm

Name

cov-preprocess Preprocess a C/C++ source file.

Synopsis

```
cov-preprocess --dir <intermediate_directory> [OPTIONS] <files>
```

Description

The `cov-preprocess` command preprocesses a source file. This command reads the command-line argument from `<intermediate_directory>/emit` and outputs the preprocessed source file or files into the `<intermediate_directory>/output/preprocessed` directory. Coverity recommends that you use the `--output-file` option to change the default output behavior whenever you need for the combined output path and filename to be an invariant (for example, in scripts) because the internal structure of the intermediate directory might change in future releases. However, note that using this option necessitates that only one file is preprocessed per invocation of `cov-preprocess`.

If you do not use an absolute path name to specify a file name, `cov-preprocess` searches for the specified file name in `<intermediate_directory>/emit`. To speed up the search time, use full path names to files that you want to preprocess.

Preprocessing expands all preprocessor directives such as `#include`, and expands macro definitions. A preprocessed source file is self-contained and can be compiled by itself with no additional files.

Note

`cov-preprocess` doesn't read source files from the emit or re-create source folders. Therefore the source files, directory structure, and working environment of the original build captured in the emit must exist and be used when `cov-preprocess` runs.

Options

--config `<coverity_config.xml>` , -c `<coverity_config.xml>`
: Use the specified configuration file instead of the default configuration file located at `<install_dir_sa>/config/coverity_config.xml`.

--diff, -d
: Preprocess the file with both the native compiler and with `cov-emit`, and then attempt to find relevant differences between the preprocessed files using some heuristics.

--diff-only
: Determine the relevant differences between two files that are already preprocessed using some heuristics.

--dir <intermediate_directory>
: Pathname to an intermediate directory that is used to store the results of the build and analysis.

-if <source_file>
: Specify an input file that is preprocessed as if it were the file that was compiled.

--native
Preprocess with the native compiler instead of with `cov-emit`.

--no-lines, -n
Do not put `#line` directives in the preprocessed output file.

--no-retranslate, -nr
Do not re-translate the command line from the original compiler when attempting to preprocess with `cov-emit`. This can be faster, but it will not work with template compiler configurations.

--output-file <output-file>, -of <output_file>
Specify the path to and file name for the output file. Coverity recommends that you use this option instead of relying on the default output behavior.

--tu <translation_unit_id(s)>, -tu <translation_unit_id(s)>
A set of translation units (TUs), named by their numeric id attribute(s). A translation unit approximately maps to the output from a single run of a compiler. This option requires a comma-separated list of id(s), and `--tu` may be specified multiple times. The union of all these identifier sets is the set of TUs to operate on subsequently, for operations that work on TUs. It is an error if any of the specified IDs do not correspond to any existing translation unit.

Shared options

--debug, -g
Turn on basic debugging output.

--info
Display certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

--ident
Display the version of Coverity Analysis and build number.

--tmpdir <tmp>, -t <tmp>
Specify the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>
Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.

- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

See Also

cov-translate

Name

cov-run-desktop Analyze locally changed files on a developer's desktop.

Synopsis

`cov-run-desktop` [OPTIONS] [FILES]

Description

The `cov-run-desktop` command performs an expedited local analysis by considering only the files or translation units specified by the user, subject to various command line options. When running `cov-run-desktop`, you must either include the specific file name(s) at the end of the command, or pass the `--analyze-scm-modified` option to let your Source Code Management system (SCM) specify which files to analyze. Translation unit selection has additional information on this process.

You can create a `coverity.conf` file to share compiler configurations and other desktop analysis settings to multiple users of the same code base. This configuration file can also be used by individual users to maintain these settings locally (see the *Coverity Desktop Analysis 8.0: User Guide* for details).

The analysis performed is the same as when using `cov-analyze` or `cov-analyze-java`, so `cov-run-desktop` accepts most of the same options as those two commands.

Because `cov-run-desktop` normally does not analyze an entire code base, it relies on summaries stored in Coverity Connect to get information about the code that is not analyzed locally. This requires that a periodic (perhaps nightly) full analysis be run in order to populate the summary information. Additionally, after the main analysis phase is complete, `cov-run-desktop` normally contacts the Coverity Connect server to determine which defects were newly introduced, and to retrieve triage data (Classification, Severity, Owner, etc.) for existing issues. `cov-run-desktop` can also operate in "disconnected" mode, relying on previously downloaded summary and issue data, if any.

By default, `cov-run-desktop` produces issue output in a text format that is intended to imitate the typical output syntax of compiler error messages. Most editors and IDEs will then automatically allow you to navigate to the corresponding locations in the source code. `cov-run-desktop` can also write the issues in JSON format so that a user-provided tool can consume and present them in a user-defined way.

The exit code of `cov-run-desktop` is 0 when the analysis completes successfully and 2 or greater if there is an error. The `--exit1-if-defects` option causes `cov-run-desktop` to exit with code 1 when defects are present.

Translation unit selection

In order to improve the speed of the analysis, `cov-run-desktop` limits the number of files or translation units considered. In this context, a translation unit refers to any primary source file from the intermediate directory, along with any files included by that primary source file. The selection process is defined using various options that affect what code will be analyzed.

There are two methods to select which translation units will be analyzed, and it is required that the user specify one. The first method is to explicitly pass a list of source files to be analyzed at the end of the command line:

```
cov-run-desktop [OPTIONS] FILE1 [FILE2 [...]]
```

For C and C++, the files will typically be `.c` or `.cpp` source files. If you want to analyze a header file, see *Coverity Desktop Analysis 8.0: User Guide*.

This can also be accomplished by listing your chosen source files in a response file, and then specifying it on the command line with the @@<response_file> syntax.

The second method of translation unit selection is to query your source code management (SCM) system for the set of files that have been modified locally and analyze those. This method is designated with the `--analyze-scm-modified` option. See `--analyze-scm-modified` and A note on Git superprojects for more information on this method.

Note

You will receive an error if you attempt to pass more than one of these methods in the same command.

A note on Git superprojects

Git superprojects are unsupported by Desktop Analysis, and will cause errors when used with `--analyze-scm-modified`.

You may be able to workaround this issue by creating a script to access Git using submodules, and specify that script with the `--scm-tool` option. This is an advanced usecase, and should only be attempted by experienced users.

Regular expressions

All `cov-run-desktop` options that call for a regular expression (`regex`) follow Perl syntax. The regular expression is case sensitive, and is considered a match if it matches a substring (i.e. full string match requires explicit anchors).

Options categories

The options accepted by `cov-run-desktop` fall into several categories:

Note

Each of the items below are linked to other sections in this document. Many link to the relevant definition in the `cov-run-desktop` section, while the analysis options link to `cov-analyze` and `cov-analyze-java`.

Options that affect what code will be analyzed

- `--analyze-scm-modified`
- `--analyze-untracked-files`
- `--ignore-modified-file-regex`

- `--ignore-modified-non-psf`
- `--ignore-untracked-file-regex`
- `--include-failed-source-only-tus`
- `--modification-date-threshold`
- `--restrict-modified-file-regex`
- `--restrict-untracked-file-regex`
- `--tu-pattern`

Options for using your Source Code Management (SCM) system

- `--analyze-untracked-files`
- `--ignore-untracked-file-regex`
- `--restrict-untracked-file-regex`
- `--scm`
- `--scm-project-root`
- `--scm-tool`
- `--scm-tool-arg`

Options that control the analysis

- `--aggressiveness-level`
- `--all`
- `--checker-option`
- `--concurrency` (C/C++)
- `--debug`
- `--debug-flags` (C/C++)
- `--disable`
- `--disable-default`
- `--disable-fb` (Java)
- `--disable-fnptr` (C/C++)

- `--disable-misra` (C/C++)
- `--disable-parse-warnings` (C/C++)
- `--disable-webapp-security` (C#, Java)
- `--distrust-all` (C#, Java)
- `--distrust-console` (C#, Java)
- `--distrust-database` (C#, Java)
- `--distrust-environment` (C#, Java)
- `--distrust-filesystem` (C#, Java)
- `--distrust-http` (C#, Java)
- `--distrust-http-header` (C#, Java)
- `--distrust-network` (C#, Java)
- `--distrust-rpc` (C#, Java)
- `--distrust-servlet` (C#, Java)
- `--distrust-system-properties` (C#, Java)
- `--enable`
- `--enable-callgraph-metrics`
- `--enable-constraint-fpp`
- `--enable-fb` (Java)
- `--enable-fnptr` (C/C++)
- `--enable-parse-warnings` (C/C++)
- `--enable-single-virtual` (C/C++)
- `--enable-virtual` (C/C++)
- `--extend-checker`
- `--extend-checker-option`
- `--fb-exclude` (Java)
- `--fb-include` (Java)

- `--fb-max-mem` (Java)
- `--fnptr-models` (C/C++)
- `--hfa` (C/C++)
- `--ident`
- `--info`
- `--inherit-taint-from-unions` (C/C++)
- `--jobs`
- `--max-loop` (C/C++)
- `--max-mem`
- `--misra-config` (C/C++)
- `--model-file`
- `--no-field-offset-escape` (C/C++)
- `--not-tainted-field` (C#, Java)
- `--override-worker-limit` (C/C++)
- `--parse-warnings-config` (C/C++)
- `--paths` (C/C++)
- `--preview` (C/C++)
- `--redirect`
- `--rule` (C/C++)
- `--security` (C/C++)
- `--security-file`
- `--strip-path`
- `--symbian` (C/C++)
- `--tainted-field` (Java)
- `--ticker-mode` (C/C++)
- `--tmpdir`
- `--trust-all` (C#, Java)

- `--trust-console` (C#, Java)
- `--trust-database` (C#, Java)
- `--trust-environment` (C#, Java)
- `--trust-filesystem` (C#, Java)
- `--trust-http` (C#, Java)
- `--trust-http-header` (C#, Java)
- `--trust-network` (C#, Java)
- `--trust-rpc` (C#, Java)
- `--trust-servlet` (C#, Java)
- `--trust-system-properties` (C#, Java)
- `--tu-pattern` (C/C++)
- `--use-reference-settings`
- [Deprecated] `--user-model-file`: Use `--model-file` instead.
- `--verbose`
- `--wait-for-license`
- `--webapp-config-checkers` (C#, Java)
- `--webapp-security-aggressiveness-level` (C#, Java)
- `--webapp-security-preview` (C#, Java)
- `--webapp-security` (C#, Java)
- `--whole-program`

Options to specify connection and triage information for the Coverity Connect server

- `--auth-key-file`
- `--certs`
- `--disconnected`
- `--host`
- `--mark-fp`

- --mark-int
- --on-new-cert
- --password
- --port
- --reference-snapshot
- --set-new-defect-owner
- --set-new-defect-owner-limit
- --set-new-defect-owner-to
- --ssl
- --stream
- --user

Output and filtering options

- --add-ignore-modified-file-regex
- --add-restrict-modified-file-regex
- --category-regex
- --checker-regex
- --cid
- --component-not-regex
- --component-regex
- --confine-to-scope
- --custom-triage-attribute-not-regex
- --custom-triage-attribute-regex
- --exit1-if-defects
- --file-regex
- --first-detected-after
- --first-detected-before

- `--function-regex`
- `--impact-regex`
- `--ignore-modified-file-regex`
- `--json-output-v[1|2|3]`
- `--kind-regex`
- `--merge-key-regex`
- `--lang`
- `--no-default-triage-filters`
- `--no-text-output`
- `--ownerLdapServerName-regex`
- `--print-path-events`
- `--present-in-reference`
- `--relative-paths`
- `--relative-to`
- `--restrict-modified-file-regex`
- `--sort`
- `--subcategory-regex`
- `--text-output`
- `--text-output-style`
- `--triage-attribute-not-regex`
- `--triage-attribute-regex`

Options

--add-ignore-modified-file-regex `<regex>`
: Specify a `<regex>` to ignore in addition to what is specified by `--ignore-modified-file-regex` and the `coverity.conf` file.

--add-restrict-modified-file-regex `<regex>`
: Specify a `<regex>` to add to those specified by `--restrict-modified-file-regex` and the `coverity.conf` file. This means that filtered translation units must match both regular expressions

specified by `--add-restrict-modified-file-regex` and `--restrict-modified-file-regex`.

--analyze-scm-modified
Specifies the translation units to be analyzed as those that have been modified locally, as referenced against your Source Code Management (SCM) system. When `--analyze-scm-modified` is passed, you must also pass the `--scm` option, or, preferably, set "`settings.scm.scm`" in `coverity.conf`.

--analyze-untracked-files <`boolean`>
When true, files reported as untracked by the SCM will be analyzed. This option is false by default.

--auth-key-file <`token`>
Specifies the file name of an authentication key that has previously been retrieved. This option has no effect in disconnected mode.

--build
Runs the build command specified in the `coverity.conf` file under `cov-build`. This is necessary so the Coverity tools know how to compile all of the source files in your project. If you add new source files or change how they are compiled, you need to re-run `cov-run-desktop --build`. This command will automatically configure compilers (`cov-run-desktop --configure`) if they have not been configured yet.

--category-regex <`regex`>
Filters the list of returned issues to include only those whose `checkerProperties` is not `null`, and where `checkerProperties.category` matches the specified regular expression.

--certs <`filename`>
In addition to CA certificates obtained from other trust stores, use the CA certificates in the given <`filename`>. For information on the new SSL certificate management functionality, please see *Coverity Platform 8.0 User and Administrator Guide*

--checker-regex <`regex`>
Filters the list of returned issues to include only those whose `checkerName` matches the specified regular expression.

--cid <`rangeFilter`>
Filters the list of returned issues to include only those whose `stateOnServer` is not `null`, and whose `stateOnServer.cid` is not `null` and matches the range filter. The range filter should match one of the following formats:

<`int`>
A single CID matching the value in <`int`>.

<`int`> "-"
A range of CIDs including <`int`> and all greater values.

"-" <`int`>
A range of CIDs including <`int`> and all lower values.

`<int1> "-" <int2>`
A range of CIDs including `<int1>`, `<int2>`, and all CID values in between.

--clean
Runs the clean command specified in the `coverity.conf` file.

--code-base-dir `<code_base_dir>`
Specifies the value of the "`code_base_dir`" variable. This is the directory that contains the `coverity.conf` file, and will generally be your SCM project root directory.

By default, `cov-run-desktop` searches upward in the file tree from where it is invoked to find a `coverity.conf` file. If one is found, then the directory containing that file is the `code_base_dir`. Otherwise, the invocation directory is the `code_base_dir`.

--component-not-regex `<regex>`
Filters the list of returned issues to include only those whose `stateOnServer` is `null` or for those which have no component names that match the specified regular expression.

--component-regex `<regex>`
Filters the list of returned issues to include only those whose `stateOnServer` is `null` or for which at least one component name matches the specified regular expression.

--config `<compilerConfigFile>`
Specifies the file name of the compiler configuration output file.

--configure
When this option is specified, `cov-run-desktop` invokes `cov-configure` on each of the `CompilerConfiguration` elements in the `coverity.conf` file.

--confine-to-scope `<boolean>`
Filters the list of returned issues to include only those whose `mainEventFilePathname` is one of the "analysis scope" files. This means that any defects found in files outside of the analysis scope will not be returned. When false, no such filtering is done.

This option is true by default.

Note

For information on how analysis scope is defined, see the *Coverity Desktop Analysis 8.0: User Guide*.

--cov-user-dir `<cov_user_dir>`
Specifies the value of the "`cov_user_dir`" variable. This corresponds to a directory where user-specific and application-specific settings are stored. By default, this is "`%APPDATA%/Coverity`" on Windows and "`$HOME/.coverity`" on unix.

--create-auth-key
Creates an authentication key file and writes it to the `auth_key_file` location specified in `coverity.conf`.

--custom-triage-attribute-not-regex <attrName> <regex>
Filters the list of returned issues to include only those whose stateOnServer is null, or whose CustomTriage does not contain a key equal to the specified attribute name, or that does contain such a key but with a corresponding value that does not match the specified regular expression.

For example, `--custom-triage-attribute-not-regex "customAttr" "customVal"` will return only those issues that *do not* have a custom attribute named "customAttr" with a value containing "customVal" as a substring.

--custom-triage-attribute-regex <attrName> <regex>
Filters the list of returned issues to include only those whose stateOnServer is null, or whose CustomTriage contains a key exactly equal to the specified attribute name and for which the corresponding value matches the specified regular expression.

For example, `--custom-triage-attribute-regex "customAttr" "customVal"` will return only those issues that have a custom attribute named "customAttr" with a value containing "customVal" as a substring.

--dir
Specifies the intermediate directory. Required if no `coverity.conf` file is present.

Note

Note that Desktop Analysis can only be run on an intermediate directory created on the same machine, and in the same source code directory, as the analysis will take place (i.e. the build and analysis processes must take place on the same machine and directory).

--disable-misra
Ignores any MISRA analysis configuration when `cov-run-desktop` uses analysis settings from the chosen reference snapshot. This is useful when the reference snapshot includes unwanted or unnecessary MISRA analysis results, but you still want to use the same settings for local analysis.

--disconnected
When specified, `cov-run-desktop` operates in "disconnected" mode. Related options will be accepted but ignored while disconnected.

--exit1-if-defects <boolean>
When true, `cov-run-desktop` exits with code 1 when defects are present in the analysis, as long as there are no errors present that cause a higher exit code. This option is false by default.

--extend-checker <executable>
This option will cause the specified `executable` to be invoked as an additional checker for Desktop Analysis. The executable can be either an absolute or relative path.

This option can be specified more than once to enable multiple Extend checkers.

--extend-checker-option <executable> <checker_name>:<option>[:<option_value>]
Passes a checker option for an Extend checker, which must be specified with `--extend-checker`. The specified `executable` must exactly match the `executable` passed to `--extend-checker`.

Example:

```
> cov-run-desktop [options] --extend-checker MY_CHECKER.exe --extend-checker-option
 MY_CHECKER.exe MY_CHECKER:option_a:true
```

--file-regex <regex>
Filters the list of returned issues to include only those whose mainEventFilePathname matches the specified regular expression.

--first-detected-after <date>
Filters the list of returned issues to include only those that were first detected after the specified date. The value of <date> must follow one of the following formats:

- YYYY-MM-DD: Specifies a year (YYYY), month (MM), and day (DD). Note that midnight in the local time zone (that is, T00:00<local>) is implicit.
- YYYY-MM-DD[T]hh:mm(:ss): Specifies a time of day along with the date. The time format accepts hours (hh), minutes (mm), and seconds (ss). Note that the local time zone is implicit.
- YYYY-MM-DD[T]hh:mm(:ss)Z: Specifies the Greenwich Mean Time (GMT) zone for the specified time and date. Here, Z refers to "Zulu", which signifies GMT.
- YYYY-MM-DD[T]hh:mm(:ss)[+-]hh:mm: Specifies the date along with an offset ([+-]) to the specified local time of day.

--first-detected-before <date>
Filters the list of returned issues to include only those that were first detected before the specified date. For the date format, see the --first-detected-after option.

--function-regex <regex>
Filters the list of returned issues to include only those whose functionDisplayName matches the specified regular expression.

--host <hostname>
Specifies the DNS hostname or IP address of the machine where Coverity Connect is running. This option is required, or must be specified in coverity.conf, unless operating in disconnected mode. If disconnected, this option has no effect.

--ignore-modified-file-regex <regex>
When specified, any file whose name matches <regex> will be treated as not modified. This option may only be specified once, but <regex> may use the "|" operator to ignore multiple files. See Translation unit selection for more information.

--ignore-modified-non-psf <boolean>
When specified as true, a translation unit will be considered modified only if its primary source file is modified. This option is false by default.

--ignore-untracked-file-regex <regex>
Untracked files are ignored if they match the <regex>.

--impact-regex <regex>
Filters the list of returned issues to include only those whose checkerProperties is not null, and where checkerProperties.impact matches the specified regular expression.

--include-failed-source-only-tus <boolean>
When true, the failed source-only translation units will be unified with the modified translation units for analysis. When false, failed source-only translation units will not be analyzed. This option is true by default.

Note

When false, a failed source-only translation unit might still be analyzed if it satisfies the conditions for a modified translation unit and matches any specified --tu-pattern.

--json-output-v1 <filename>, --json-output-v2 <filename>, --json-output-v3 <filename>
When present, cov-run-desktop's output is written in JSON output to the file specified in <filename>.

--json-output-v3 is the recommended JSON output option, as it contains the most complete set of information. json-output-v1 and v2 are supported for backward compatibility.

--kind-regex <regex>
Filters the list of returned issues to include only those whose checkerProperties is not null, and where the issueKinds value matches the specified regular expression.

Matching is done using a single string, which is a comma-separated concatenation of all the issueKinds, in alphabetical order. For example, "quality,security".

--lang <language>
Write event messages in the specified language. Currently, the supported values are en (for English) and ja (for Japanese). The default language is English (en).

--mark-fp <cid> <explanation>
Sets the Classification of the specified CID to "False Positive". The value of <cid> is the defect's CID, and <explanation> is a string which explains why this defect is a False Positive.

--mark-int <cid> <explanation>
Sets the Classification of the specified CID to "Intentional". The value of <cid> is the defect's CID, and <explanation> is a string which explains why this defect is Intentional.

--merge-key-regex <regex>
Filters the list of returned issues to include only those whose mergeKey matches the specified regular expression.

--modification-date-threshold <date_time>
Specifies the modification date threshold to use instead of the default. Only those files modified on or after the specified date will be included in the analysis. The value of <date_time> should match the YYYY-MM-DD[T]hh:mm(:ss) format, where date and time are separated by a space or T. The time, specified by hh:mm:ss, is optional, and seconds (:ss) are not required. If time is not specified, the default value is midnight (00:00) of the specified date.

--no-default-triage-filters
By default, `cov-run-desktop` has three active issue filters, which have the effect of suppressing issues triaged as uninteresting, or in a component named to hold third-party code:

- `--component-not-regex "[Tt]hird.*[Pp]arty"`
- `--triage-attribute-not-regex "classification" \`
  `"False Positive|Intentional|No Test Needed|Tested Elsewhere"`
- `--triage-attribute-not-regex "action" "Ignore"`

If `--no-default-triage-filters` is specified, then all three of these filters are deactivated. If `--component-regex` or `--component-not-regex` is specified, then the first filter is deactivated. If `--triage-attribute-regex` or `--triage-attribute-not-regex` is specified for "classification" or "action", then the respective filter for that attribute is deactivated.

--no-text-output
When present, `cov-run-desktop` does not print the compiler-like textual output.

--on-new-cert `<trust|distrust>`
Indicates whether to trust (with trust-first-time) self-signed certificates, presented by the server, that haven't been seen before. For information on the new SSL certificate management functionality, please see *Coverity Platform 8.0 User and Administrator Guide*

--ownerLdapServerName-regex `<regex>`
Filters the list of returned issues to include only those whose `stateOnServer` is `null`, or whose `ownerLdapServerName` matches the specified regular expression.

--password `<password>`
Specifies the password for connecting to the Coverity Connect server.

--port `<port_number>`
Specifies the HTTP or HTTPS port of the Coverity Connect server. The default value is 8080. If `--ssl` is present, the default value is 8443. This option has no effect in disconnected mode.

--print-path-events `<boolean>`
When true, path events are printed in the text output. When false, path events are not printed. This option is true by default.

--present-in-reference `<boolean>`
Filters the list of returned issues to include only those whose `stateOnServer` is `null`, or whose value for `presentInReferenceSnapshot` is equal to the specified boolean.

--reference-snapshot `<specification>`
Specifies how `cov-run-desktop` should select a reference snapshot from the selected stream. The `<specification>` must be one of the following values:

id:`<ID>`
Use the snapshot with the matching `<ID>`. `--reference-snapshot` will return an error if the ID is invalid or if it is not in the selected stream.

date:<date_time>
Use the snapshot that was created closest to, but not after, the specified <date_time>. The value of <date_time> should match the YYYY-MM-DD[T]hh:mm(:ss) format, where date and time are separated by a space or T. The time, specified by hh:mm:ss, is optional, and seconds (:ss) are not required.

--reference-snapshot will return an error if there is no snapshot with summary data that was created before the specified <date_time>.

latest
Use the latest snapshot with summary data in the specified stream.

idir-date
Use the snapshot created closest to, but not after, the creation date of the intermediate directory.

This is the default option.

scm
This option will query the SCM to determine the version that was most recently checked out or updated, and then use the closest snapshot.

The --scm option, or the "settings.scm.scm" attribute in coverity.conf, is required when using this specification.

--relative-paths <boolean>
When true, compiler-like output paths are printed as relative paths, relative to the directory specified by --relative-to. If --relative-to is not specified, the current working directory is used.

--relative-to <path>
When --relative-paths is true, compiler-like output paths are printed relative to the specified <path>. If not specified, the current working directory is used.

@@<response_file>
Specify a response file that contains a list of additional command line arguments, such as a list of files for analysis. Each line in the file is treated as one argument, regardless of spaces, quotes, etc. The file is read using the platform default character encoding.

Optionally, you can choose a different encoding, by specifying it after the first "@". For example:

```
cov-run-dekstop [OPTIONS] @UTF-16@my_response_file.txt
```

You must use a supported Coverity encoding, listed under the cov-build --encoding option.

--restrict-modified-file-regex <regex>
When specified, only files whose name matches <regex> (and whose timestamp satisfies the modification date threshold) will be included in the analysis. This option may only be specified once, but <regex> may use the "|" operator to include multiple files.

Note

--ignore-modified-file-regex takes precedence if used in tandem with --restrict-modified-file-regex.

--restrict-untracked-file-regex <regex>
Only untracked files that match the <regex>, and do not match `--ignore-untracked-file-regex`, will be analyzed.

--scm <scm_type>
Specifies the name of the source control management system. For this option to work, the command requires that source files remain in their usual locations in the checked-out source tree. If the files are copied to a different location after checkout, the SCM query will not work.

The supported scm tools are:

- `accurev`
- `clearcase`
- `cvs`
- `git`
- `perforce`
- `perforce2009`
- `svn`

On Windows, the tools `tfs2008`, `tfs2010`, `tfs2012`, `tfs2013`, and `tfs2015` are also supported.

This option behaves the same as `cov-extract-scm --scm`. See cov-extract-scm for additional details.

--scm-project-root <scm_root_path>
Specifies a path that represents the root of the source control repository. Use this option when `cov-run-desktop` is being run from a directory other than the root of the source control repository. All paths returned by `--get-modified-files` will be relative to this path.

This option behaves the same as `cov-extract-scm --project-root`. See cov-extract-scm for additional details.

--scm-tool <scm_tool_path>
Specifies the path to an executable that interacts with the source control repository. If the executable name is given, it is assumed that it can be found in the path environment variable. If not provided, the command uses the default tool for the specified `--scm` system.

This option behaves the same as `cov-extract-scm --tool`. See cov-extract-scm for additional details.

--scm-tool-arg <scm_root_path>
Specifies additional arguments that are passed to the SCM tool, specified in the `--tool` option, that gathers the last modified dates. The arguments are placed before the command and after the tool. This option can be specified multiple times.

This option behaves the same as `cov-extract-scm --tool-arg`. See `cov-extract-scm` for additional details.

--set-new-defect-owner <boolean>
When true, and in connected mode, sets the owner for newly detected defects that exist locally as the current user. True by default.

See also, `--set-new-defect-owner-limit`.

--set-new-defect-owner-limit <limit>
Set the limit on the number of defects to assign to the current user. If the number of discovered defects is more than the limit, then skip the assignment. The default limit is 100.

Note

`--set-new-defect-owner` and `--set-new-defect-owner-limit` have no effect on the following platforms:

- FreeBSD
- HP-UX
- Itanium
- NetBSD

--set-new-defect-owner-to <user>
When used with `--set-new-defect-owner`, this specifies the user to whom any new defects will be assigned. The default is the current user.

Note that the specified <user> must already exist in the Coverity Connect database.

--setup
This option is intended as a single step to get a new user ready for Desktop Analysis. It creates an authentication key (if one has not already been created), runs the clean command, and then captures a full build.

--sort <sort_spec>
Specifies the sort order for text output. The <sort-spec> accepts the values listed below. To sort on more than one attribute, you can use a non-empty, comma-separated list of values. Additionally, to specify ascending or descending sort order for any attribute, you can add `:a` or `:d` (respectively) directly after the attribute name. All attributes, except `cid`, are ordered in ascending order by default.

The available sort attributes are:

- `cid`: ID number assigned by Coverity Connect to each issue.
- `occurrence`: The number of the occurrence among all those in the output set that have the same merge key.
- `occurrences`: The total number of issue occurrences.

- `mergeKey`: An internal identifier used to assign CIDs to issues. This is mainly useful when the CID is missing, either because `cov-run-desktop` is in disconnected mode or because the Coverity Connect server is a subscriber that is disconnected from its coordinator.
- `checker`: The name of the checker that found this issue.
- `file`: The complete path to the file that contains the issue.
- `line`: The file's line number where the issue is located.
- `function`: The name of the function that contains the issue.
- `impact`: The issue's impact, as determined by Coverity Connect: High, Medium, or Low
- `category`: Description of the nature of the software issue.
- `subcategory`: The sub-category of the defect reported by `<checker>`.
- `present`: True if the issue is present in the specified snapshot, otherwise false.
- `ownerLdapServerName`: The LDAP server of the defect owner.
- `component`: The name of the component that contains this issue.
- `firstDetected`: The date and time in which the issue was first detected by the analysis.
- `classification`: The value of the issue's classification attribute.
- `action`: The specified action to be taken on the issue.
- `fixTarget`: Target milestone for fixing an issue.
- `severity`: The value of the issue's severity attribute.
- `legacy`: True if the issue is marked as a `legacy` issue, otherwise false.
- `owner`: The user assigned to the issue.
- `externalReference`: An internal identifier used by your company to track the issue.
- `customTriage[<attribute>]`: The value of any custom triage attributes. Within the bracket-enclosed `<attribute>` name, use two consecutive right brackets (`]]`) to encode a single right bracket (`]`).

For example, `--sort file,classification:d` will order the results first by ascending file name, then descending Classification.

--ssl
: When present, use SSL encryption for all communication with Coverity Connect. This option has no effect in disconnected mode.

--stream `<stream_name>`
: Specifies the Coverity Connect stream which contains the relevant snapshot and triage information. This option has no effect in disconnected mode.

--subcategory-regex <regex>
Filters the list of returned issues to include only those whose checkerProperties is not null, and where checkerProperties.subcategoryShortDescription matches the specified regular expression.

--text-output <filename>
Write the text output to the specified file rather than writing it to the console.

--text-output-style <style>
Specifies the style of the text output. There are two accepted styles:

- oneline - Each occurrence and event is written as a single line of output. This format works best with vi type editors.
- multiline - Each occurrence and event is split accross multiple lines. This format works best with Emacs editors.
- msvs - Similar to multiline, but prints locations as <file(line)> instead of <file:line>; for use with Visual Studio.

If unspecified, the multiline style is used by default.

--triage-attribute-not-regex <attrName> <regex>
Filters the list of returned issues to include only those whose stateOnServer is null, or whose Triage does not contain a key equal to the specified attribute name, or that does contain such a key but with a corresponding value that does not match the specified regular expression.

For example, --triage-attribute-not-regex "severity" "Minor" will return only those issues that *do not* have a "severity" attribute with a value containing "Minor" as a substring.

--triage-attribute-regex <attrName> <regex>
Filters the list of returned issues to include only those whose stateOnServer is null, or whose Triage contains a key exactly equal to the specified attribute name and for which the corresponding value matches the specified regular expression.

For example, --triage-attribute-regex "classification" "Bug" will return only those issues that have a "classification" attribute with the value, "Bug" (or containing the substring "Bug").

--tu-pattern <pattern>
When specified, only those translation units which match <pattern> will be present in the analysis.

--upgrade <version>
Launches an assisted upgrade of Coverity Analysis to the version specified. The value of <version> should be the desired Coverity Analysis version number (8.0.0 for example). This will download the required installer from Coverity Connect, run it, and provide instructions to invoke the new version. The existing installation with *not* be overwritten.

--use-reference-settings <boolean>
If true, cov-run-desktop will use the analysis settings downloaded from the reference snapshot. True by default.

--user <user_name>
Specifies the Coverity Connect user name. If unspecified, the default is the value for the environment variable, "COV_USER", "USER", or "USERNAME", if specified. If none of these is specified, and "settings.server.username" is not specified in the coverity.conf file, then the --user option is required.

This option has no effect in disconnected mode.

--whole-program
Some checkers, such as Application Security checkers, are only effective when analyzing all source files in the program. By default, these "whole-program" checkers are not available to cov-run-desktop.

Specifying --whole-program allows cov-run-desktop to run whole-program checkers.

Example 1

```
> cov-run-desktop file1.c file2.c
```

This will analyze file1.c and file2.c, assuming a compilation of those files has previously been captured using cov-build.

Example 2

```
> cov-run-desktop --analyze-scm-modified
```

This will query your SCM to find out which files have been modified locally and analyze those.

Example 3

```
> cov-run-desktop --dir idir --host my_server --port 8080 \
        --auth-key-file keyfile --stream my_stream \
        --triage-attribute-regex Owner user1 \
        --text-output-style oneline \
        file1.c file2.c
```

This example analyzes both file1.c and file2.c, obtains a reference snapshot from the "my_stream" stream on my_server:8080, authenticates using "keyfile" (which must have been previously created using cov-run-desktop), filters the results for those assigned to user1, and writes the defects to the console in the oneline format, which uses one line of text for each defect and event. Notice that many of the options specified here could instead have been put into a coverity.conf file for convenience.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.

- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

See Also

cov-build

cov-configure

cov-manage-im

cov-manage-emit

Name

cov-test-configuration Test command-line translations of a configuration by making assertions about the translations.

Synopsis

```
cov-test-configuration [OPTIONS] <input_script.json>
```

Description

The `cov-test-configuration` command parses the given input script, invokes `cov-translate` on the native compiler commands, and asserts that the given facts about the translated command lines are true. The translation takes place according to the configuration generated by the `cov-configure` command.

If all tests pass, it exits with 0. Otherwise, it exits with a non-zero code.

Note

The `cov-translate` command uses the same configuration as this command and translates native command-line options into `cov-emit` options but does *NOT* execute `cov-emit`. See `cov-translate --dry-run`.

Any command names referenced by the input script must be executable in the current environment.

Input script format. The input script must be a well-formed, valid JSON text file encoded in UTF-8. See www.json.org for more information on JSON.

Sample input script format:

```
[
{
 "section": "My Section Label",
 "tests": [
   {
     "given_input" : "gcc -m64 -c foo.c",
     "assert_that_output": {
       "contains": [ "-w", "-D__NO_INLINE__", "--incompat_proto" ],
       "omits": [ "-m32" ]
     }
   }
   , ... // more tests here
 ]
}
, ... // more sections here
]
```

The expected structure is the following:

- A list of one or more `section` objects.

- Each `section` object is composed of:
  - section: A label naming the section of tests.
  - tests: A list of one or more `test` objects.
- Each `test` object is composed of:
  - `given_input`: The native command line to translate.
  - `assert_that_output`: An `assertion` object applied to translated command line.
- An `assertion` object is composed of one or more of the following `assertion` operators:
  - `contains`: List of one or more command-line options that are required in translation.
  - `omits`: List of one or more command-line options that are forbidden in translation.

Each test translates the `given_input` with `cov-translate` and checks that all the `assertion` operators pass when applied to the translation.

Example 1:

```
> cov-configure --config myTest/coverity_config.xml --msvc
> cov-test-configuration --config myTest/coverity_config.xml MyTests.json
```

Output of the `cov-test-configuration` example:

```
Section [0] My Section Label
Tests run: 1, Failures: 0, Errors: 0

Sections run: 1, Tests run: 1, Failures: 0, Errors: 0
```

In this example, all tests passed.

Example 2:

```
> cov-test-configuration --config myTest/coverity_config.xml OtherTests.json
```

Output of the `cov-test-configuration` example:

```
Section [0] My Section Label
Tests run: 1, Failures: 0, Errors: 0

Section [1] Microsoft C/C++
Tests run: 5, Failures: 2, Errors: 1
```

```
Section [1] Microsoft C/C++: Test [0]: Assertion [contains][2] failed
Given input: cl -c foo.c
Expected   : output contains bob
Actual     : cov-emit.exe ... --c --microsoft --no_alternative_tokens \
-w --ignore_calling_convention --microsoft_version 1300 \
--no_stdarg_builtin -D_USE_ATTRIBUTES_FOR_SAL=0 foo.c

Sections run: 2, Tests run: 6, Failures: 2, Errors: 0
```

The example shows two test failures. Failures are `assertion` operators that failed. Errors are failures that were encountered when executing `cov-translate`.

The ellipsis (`...`) in the example represents other arguments that `cov-translate` adds to make `cov-emit` imitate the `cl` input more precisely.

Shared options

--config `<coverity_config.xml>` , -c `<coverity_config.xml>`
Use the specified configuration file instead of the default configuration file located at `<install_dir_sa>/config/coverity_config.xml`.

--debug, -g
Turn on basic debugging output.

--ident
Display the version of Coverity Analysis and build number.

--info
Display certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

--verbose `<0, 1, 2, 3, 4>`, -V `<0, 1, 2, 3, 4>`
Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

See Also

cov-configure

cov-emit

cov-translate

Name

cov-translate Translate native compiler command line arguments (for C/C++) into `cov-emit` arguments.

Synopsis

```
cov-translate --dir <intermediate_directory> [OPTIONS] <compile-command>
```

Description

Translate the native compiler command line given and invoke the `cov-emit` command with a translation of the command-line arguments. The translation is done according to the configuration generated by the `cov-configure` command.

Note

If you change the set of compilation options used by your build process, delete the `<intermediate_directory>` and capture a full build from scratch. Otherwise, the translation units captured using the old options will remain in the emit. The new translation units will not replace the old translation units due to the changed compilation options.

Example

Suppose you have a single file `t.c` that is compiled with gcc as:

```
> gcc -DFOO=BAR -c t.c
```

In this case, invoking this command with `cov-translate` in front will call `cov-emit` in the appropriate way to compile `t.c`, assuming that gcc has been configured with `cov-configure`:

```
> cov-translate --dir /tmp/emit gcc -DFOO=BAR -c t.c
```

Output of the `cov-translate` example:

```
    cov-emit --dir /tmp/emit ... --gcc -w -DFOO=BAR t.c
    Emit for file 't.c' complete.
```

In the previous example, `...` represents other arguments that are added to provide system include directories, preinclude files, and other command-line arguments that `cov-translate` adds to make `cov-emit` imitate gcc more precisely.

Options

--add-arg <arg>
: Directly add `<arg>` to the argument list passed to `cov-emit`. These arguments are passed before other arguments detected by `cov-translate`.

--add-arg-no-file <arg>
: Add an argument `<arg>` to the invocations of `cov-emit`. This argument is not stored in the response file, and therefore is not used by `cov-build --replay` or `cov-preprocess`.

--auto-diff
Turn on automatic diagnosis of parsing problems by comparing preprocessed files generated by the native compiler versus `cov-emit`. It can be useful to determine incompatibilities (especially of include order and macro definitions), but it might interfere with the build for some compilers that are not fully supported.

--chase-symlinks
Follow symbolic links when determining file names to report.

This option is not supported for use with Clang compilers.

--clean-preprocessed
Clean up preprocessed files left behind by `--preprocess-first` after each file is compiled. Usually `--preprocess-first` leaves preprocessed files in the source code directory. This option only makes sense if `--preprocessed-first` is also specified.

--cygpath <path>
Specify the path to the directory, which contains the bin directory of the Cygwin installation, if it is not in the PATH environment variable.

--cygwin
On Windows, indicates that the build is done with Cygwin. This option allows Cygwin-style paths to be used in the native build command. However, you must use Windows-style paths for all Coverity Analysis commands.

--dir <intermediate_dir>
Pathname to an intermediate directory that is used to store the emit repository and output directory.

Unless you are running `cov-translate` for debugging purposes, you should always use the `--dir` option. Without it, the command will not emit output an intermediate directory.

Note that you can specify this pathname in `coverity_config.xml`. For details on using this file, see on the `<prevent>` tag in the section "Using the <prevent> tag to specify directories and emit options" in *Coverity Analysis 8.0 User and Administrator Guide*.

--dryrun, -n
Prints out the `cov-emit` command line it would normally run without actually running it. This option can help expedite your configuration process. The option is analogous to `make -n`.

--emit-cmd-line-id
This option is deprecated as of the 4.4 release.

--emit-parse-errors
Deprecated. Use the `--enable PARSE_ERROR` option in `cov-analyze` instead. Specifies that compile failures should be made visible in Coverity Connect, appearing as defects.

--emulate-string <regex>, -s <regex>
Specify a regex that, if matched on the command line, causes cov-translate to only run the native compiler command line given without attempting to call `cov-emit`. This is useful for files such as conftest.c, which are compiled by "configure" to test the native compiler's output for certain strings.

Having cov-translate's output interspersed with these strings causes configure to fail. This option only makes sense if `--run-compile` is also specified.

--encoding <enc>
Specifies the encoding of source files. Use this option when the source code contains non-ASCII characters so that Coverity Connect can display the code correctly. The default value is US-ASCII. Valid values are the ICU-supported encoding names:

US-ASCII

UTF-8

UTF-16

UTF-16BE
UTF-16 Big-Endian

UTF-16LE
UTF-16 Little-Endian

UTF-32

UTF-32BE
UTF-32 Big-Endian

UTF-32LE
UTF-32 Little-Endian

ISO-8859-1
Western European (Latin-1)

ISO-8859-2
Central European

ISO-8859-3
Maltese, Esperanto

ISO-8859-4
North European

ISO-8859-5
Cyrillic

ISO-8859-6
Arabic

ISO-8859-7
Greek

ISO-8859-8
Hebrew

ISO-8859-9
: Turkish

ISO-8859-10
: Nordic

ISO-8859-13
: Baltic Rim

ISO-8859-15
: Latin-9

Shift_JIS
: Japanese

EUC-JP
: Japanese

ISO-2022-JP
: Japanese

GB2312
: Chinese (EUC-CN)

ISO-2022-CN
: Simplified Chinese

Big5
: Traditional Chinese

EUC-TW
: Taiwanese

EUC-KR
: Korean

ISO-2022-KR
: Korean

KOI8-R
: Russian

windows-1251
: Windows Cyrillic

windows-1252
: Windows Latin-1

windows-1256
: Windows Arabic

-E
Only preprocess the source file.

--fail-stop, -fs
Return an error code if `cov-emit` fails. By default, `cov-emit` parse failures are ignored and the return code is success.

--force
Passes `--force` to `cov-emit`, which causes emits to happen for files whose timestamps have not changed. See `--force` in `cov-emit`.

--no-caa-info
Do not collect the information required for Coverity Architecture Analysis in the intermediate directory.

--no-emit
Parse, but do not emit, source files. This is useful primarily for debugging purposes and is not intended for general use.

--no-headers
Do not emit header files, only the primary source file. Because this option can cause problems in C++ programs, you should use it only if directed by Coverity support.

--no-parallel-translate
Disables `cov-translate` parallelization. This will prevent `cov-translate` from running in parallel regardless of the degree of parallelization requested, either directly to `cov-build`, `cov-translate`, through configuration files, or native command line translation.

This can also be added as a `cov-emit` argument in a configuration file (it is not actually passed to `cov-emit`). For example:

```
<prepend_arg>--no-parallel-translate</prepend_arg>
```

--no-preprocess-next
Disables the `--preprocess-next` option.

--parallel-translate=<number_of_processes>
Compatible with C and C++ builds only. Instructs `cov-translate` to run `cov-emit` in parallel when multiple files are seen on a single native compiler invocation. This is similar to the Microsoft Visual C/C++ `/MP` switch. Specify the <number_of_processes> to be greater than zero to explicitly set the number of processes to spawn in parallel, or zero to auto-detect based on the number of CPUs. When specified directly to `cov-build` or `cov-translate`, this option will override any settings set in configuration files or translated through the native command line.

This can also be added as a `cov-emit` argument in a configuration file (it is not actually passed to `cov-emit`). For example:

```
<prepend_arg>--parallel-translate=4</prepend_arg>
```

--preinclude <file.h>, -pi <file.h>
Specify that <file.h> should be preincluded before all other source and header files when invoking `cov-emit`. This is equivalent to `cov-emit`'s `--preinclude` argument.

--preprocess-first
Compatible with C and C++ builds only. Uses the native compiler to preprocess source files and then invokes `cov-emit` to compile the output of the native processor. By default, `cov-emit` (which is invoked by `cov-translate`) otherwise tries to preprocess and parse each source file.

Using this option can address some cases in which hard-to-diagnose causes for macro predefinitions are different, or for header files that cannot be found by `cov-emit`. Usually, `cov-configure` attempts to intelligently guess the native compiler's predefined macros and built-in include directories, but sometimes `cov-configure` guesses incorrectly. Using the `--preprocess-first` option circumvents the problem, but at the cost of losing macro information during analysis. Using `--preprocess-first` does not always work because it requires rewriting the native compiler command line, which the native compiler may or may not like.

--preprocess-next
Compatible with C and C++ builds only. Attempts to use `cov-emit` to preprocess source files. If that attempt fails, or if `cov-emit` encounters a parse error, this option preprocesses the files with the native preprocessor, and invokes `cov-emit` to compile the output of the native processor. This offers the benefit of using the higher-fidelity `cov-emit` preprocessor, while also providing a fallback in case of errors.

This option can be disabled with --no-preprocess-next (the latter has precedence over the former). See `--preprocess-first` for information about the effects of using the native preprocessor.

--print-native
Print out the native compiler command line.

--record-only, -ro
Only record the compilation command in the emit directory, but do not attempt to parse and emit the code yet. Later, `cov-build` can be run with --replay to actually parse and emit the code.

--redirect stdout|stderr,<filename>, -rd stdout|stderr,<filename>
Redirect either stdout or stderr to <filename>..

--run-compile
In addition to calling `cov-emit`, also run the native compiler. This should only be used if you are attempting to manually integrate cov-translate into your build system; it should never be used with `cov-build`.

--timings
Pass -# to `cov-emit`, which prints out timing information for various stages of parsing and emitting for every file compiled. This is a debugging option not intended for general use.

Shared options

--config `<coverity_config.xml>` , -c `<coverity_config.xml>`
Use the specified configuration file instead of the default configuration file located at `<install_dir_sa>/config/coverity_config.xml`.

--debug, -g
Turn on basic debugging output.

--debug-flags <flag> [, <flag>]
: Controls the amount of debugging output produced during a build. These flags can be combined on the command line using a comma as a delimiter.

  Valid flags are `translate` and `translate-phases`. For example, `--debug-flags translate`.

--ident
: Display the version of Coverity Analysis and build number.

--info
: Display certain internal information (useful for debugging), including the temporary directory, user name and host name, and process ID.

--tmpdir <tmp>, -t <tmp>
: Specify the temporary directory to use. On UNIX, the default is `$TMPDIR`, or `/tmp` if that variable does not exist. On Windows, the default is to use the temporary directory specified by the operating system.

--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>
: Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

See Also

cov-build

cov-configure

cov-emit

cov-link

Name

cov-upgrade-static-analysis Upgrade an old Coverity Analysis release to the latest version.

Synopsis

`cov-upgrade-static-analysis` { --use-existing-release | --use-new-release } --old-release <dir> [--old-config <file>...]
[OTHER OPTIONS]

Description

The `cov-upgrade-static-analysis` command upgrades an old Static Analysis or Coverity Analysis release to Coverity Analysis version 8.0. Run this command from the `<install_dir_sa>/bin` directory of the new release of Coverity Analysis. Also, the exact upgrade process differs slightly based on file permission and web server process owner issues.

There are two modes in which you can run this command.

- In the first (and preferred) mode of operation, specified with the `--use-new-release` option, the configuration and the database in the old release is moved into the new release.
- In the second mode, specified with the `--use-existing-release` option, the old release is upgraded in place. When the upgrade is completed, the new release is installed in the location that was formerly occupied by the old release.

Options

Basic options

--old-config <file>
: The location of an old Coverity Analysis non-default configuration file (`coverity_config.xml`) that for the old release. Specifying the path to all configuration files referenced on a typical command line to the old release allows the upgrade to re-write any configuration files that are relevant when invoking the programs in the new release. Repeat this option for each configuration file.

--old-release <dir>, -or <dir>
: The location of the old Coverity Analysis release. This should be the full path to the directory containing the version file, which may be the textual `VERSION` file, the XML `VERSION.xml` file, or both.

--use-existing-release
: Upgrade an existing Coverity Analysis release in-place.

  Do not use this option if you are upgrading to the current version of Coverity Analysis .

--use-new-release
: Copy settings and the database from the old Coverity Analysis release to the new release. Leave the old release untouched. Note that any files in the old release that were added by the user are added to the new release automatically when the upgrade is complete. This is the preferred mode of operation.

Other options

--help, -h
Provide help information on the command.

--log `<file>`
The absolute path and file name for where to save the upgrade log. The default log file is `<install_dir_sa>/bin/coverity_upgrade.log`.

--pedantic
If the command returns with warnings, return a non-zero exit code. The default behavior is to return non-zero codes only when there are errors.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

Name

cov-wizard Launches a GUI-based utility for configuring Coverity Analysis.

Synopsis

`cov-wizard <command>`

Description

The `cov-wizard` command runs a GUI-based Coverity Analysis utility for setting up compilers, configuring and running the build process, running the analysis for quality and security issues, and committing the results to Coverity Connect. Java code and C/C++ code. For information about Coverity Wizard, see the *Coverity Wizard 8.0 User Guide* .

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

Coverity Analysis Ant Tasks

Name

covanalyzeandcommit Analyze Java source code and class files, and then commit the results to Coverity Connect.

Synopsis

```
<covanalyzeandcommit
  dir="int_dir"
  [OPTIONAL_ATTRIBUTES]>
    <[OPTIONAL_ELEMENTS]/>
</covanalyzeandcommit>
```

Description

The `covanalyzeandcommit` analyzes Java source code and class files that have been previously stored in an intermediate directory by capturing a build with the `cov-build` command and/or by adding them manually using the `cov-emit-java` command. Then it commits the results to Coverity Connect. The commit process occurs if the analysis had either no errors or only recoverable errors. Otherwise, the commit process is skipped.

Attributes

additionalanalysisoptions="options"

Most users are unlikely to use this attribute, which provides a string of additional, space-separated options to pass to `cov-analyze-java`. You might use it to pass options that are not available as analysis-related attributes to this Ant task.

This attribute should not be used with any options that contain spaces, such as filenames with spaces.

Example:

```
<covanalyzeandcommit additionalanalysisoptions="--append false --max-mem 8000"/>
```

See `cov-analyze-java` for a complete list of options.

additionalcommitoptions="options"

Most users are unlikely to use this attribute, which provides a string of additional, space-separated options to pass to `cov-commit-defects`. You might use it to pass options that are not available as commit-related attributes to this Ant task.

This attribute should not be used with any options that contain spaces, such as filenames with spaces.

Example:

```
<covanalyzeandcommit additionalcommitoptions="--debug false --ticker-mode none"/>
```

See `cov-commit-defects` for a complete list of options.

all="true"
Enables Coverity Analysis for Java checkers that are disabled by default.

Exception: Web application security checkers (such as XSS) are not affected by this option. To enable them, see `--webapp-security`, `--webapp-config-checkers`, and `--webapp-security-preview`.

analysis="false"
Allows you to turn off the analysis process, which normally takes place prior to the commit process. In this way, you can commit the results of a prior analysis to Coverity Connect without running another analysis first. Defaults to true.

binpath="`<install_dir>`/bin"
Specifies the directory containing `cov-analyze-java` and `cov-commit-defects`. Use this attribute if the Ant task fails to find these commands. Without this attribute, the Ant task searches for these based on the PATH environment variable and/or the location of `coverity-anttask.jar`.

commit="false"
Allows you to turn off the commit process. In this way, you can perform an analysis without committing results to Coverity Connect afterward. Defaults to true.

config="coverity_config.xml"
Uses the specified configuration file instead of the default configuration file located at `<install_dir>/config/coverity_config.xml`. This file applies to both the analysis and the commit processes.

dataport="cim_commit_port"
Specifies the commit port of the Coverity Connect server.

dir="int_dir"
Pathname to an intermediate directory that is used to store the emit repository and output directory.

If you specify ".", it uses the current directory as the intermediate directory.

disabledefault="true"
Disables default checkers. This option is useful if you want to disable all default checkers and then enable only a few with the `--enable` option.

For a list of checkers that are disabled through this option, see the `--enable` option documentation for the `cov-analyze` command.

failonerror="true|false"
If true, the build process will succeed only if both `cov-analyze-java` and `cov-commit-defects` exit without return codes that indicate failure. Otherwise, the Ant task will always succeed. Defaults to `true`.

findbugs="true|false"
Explicitly enables (`true`) or disables (`false`) FindBugs™ analysis. This attribute corresponds to the `enable-fb` (when true) and `disable-fb` (when false) options to `cov-analyze-java`. FindBugs

analysis is enabled by default. Further, you can use the `disable checker` element to disable individual FindBugs bug patterns. However, `enable checker` *will not* enable them.

This attribute supports the analysis process.

host="server_hostname"
Specify the server hostname to which to send the results. The server must have a running instance of Coverity Connect.

If unspecified, the default is the `host` element from the XML configuration file.

Note

If you're running `cov-commit-defects` on a Linux OS, or using `--ssl`, you must enter the full host and domain name for the `--host` parameter:

```
--host <server_hostname.domain.com>
```

httpport="port_number"
Used with the `--host` option to specify the HTTP port on the Coverity Connect host. This port is used to connect to the `--dataport` commit port.

The commit port is determined using one of the following methods, listed in order of priority (the first applicable item will be used):

1. The commit port specified with `--dataport.`
2. The HTTP port specified with `--port`. `cov-commit-defects` connects to this port to retrieve the dataport using HTTP if `--ssl` is absent or HTTPS if `--ssl` is present.
3. The HTTPS port specified with `--https-port`. `cov-commit-defects` connects to this port using HTTPS to retrieve the dataport.
4. The commit port, specified with the `cim/commit/port` element from the XML configuration file.
5. The HTTP or HTTPS port specified with the `cim/port` or `cim/https_port` element, respectively, from the XML configuration file.
6. HTTP port 8080 without `--ssl` or 8443 with `--ssl` is used to retrieve the dataport from Coverity Connect.

Note

If you are committing to an SSL-enabled instance of Coverity Connect, you might encounter an error message when you define the `--port option` (for example, `--port 8443.` Use the `--https-port` option instead.

parallelthreads="N"
Allows you to control the number of analysis workers that run in parallel. This number is limited by the terms of your license. The default value for the `-j` option is 1.

This attribute supports the analysis process.

password="password"
Specify the password for either the current user name, or the user specified with the `--user` option. For security reasons, the password transmitted to the Coverity Connect is encrypted. If unspecified, the default is (in order of precedence):

1. The `password` element from the XML configuration file.
2. The environment variable `COVERITY_PASSPHRASE`.
3. The password in the file pointed to by the environment variable `COVERITY_PASSPHRASE_FILE`.

Note

The passphrase can be stored in a file without any other text, such as a newline character.

Warning

On multi-user systems, such as Linux, users can see the full command line of all commands that all users execute. For example, if a user uses the `ps -Awf` command, identifying information such as usernames, process identities, dates and times, and full command lines display.

This attribute supports the commit process.

resultproperty="property_name"
The name of a property to store the return code of the command. It provides the maximum value of the `cov-analyze-java` and `cov-commit-defects` return code. Only used if `failonerror=false`.

This attribute supports the analysis process.

stream="stream_name"
Specifies a stream name to which to commit these defects.

If the stream option is not specified, the `stream` element from the XML configuration file is used.

If the stream is associated with a specific language and you attempt to commit results from other languages to that stream, the commit will fail. However, in Coverity Connect, it is possible to associate a stream with multiple languages even if the stream was previously associated with a single programming language.

strippath="path"
Strip the prefix `<path>` from all file names in error messages and file references committed. This might make commits from multiple users match, even if the code is located in a different location.

If specified multiple times, strips all of the prefixes from each filename, in the order the `--strip-path` arguments are supplied.

This attribute supports the commit process.

target="target_name"
Target platform for this project (for example, `i386`).

user="user_name"
Specifies the user name that is shown in Coverity Connect as having committed this snapshot. If unspecified, the default is:

1. The `user` element from the XML configuration file.
2. The environment variable `COV_USER`.
3. The environment variable `USER`.
4. The name of the operating system user invoking the command (where supported).
5. The UID of the operating system user invoking the command (where supported).
6. `admin`.

version="version"
This snapshot's project version.

Elements

The following elements must be enclosed by the `covanalyzejava` element.

`<checkeroption checker="checker_name" option="option_name" val="value"/>`
Pass option `option_name` (with optional value `value`) to a specific checker `checker_name`.

For example:

```
<checkeroption checker="NULL_RETURNS" option="check-bias" value="5"/>
```

`<disable checker="checker_name"/>`
Disable `checker_name`. This can be specified multiple times. See also `--list-checkers` and `--disable-default`. For example, to disable cross-references in defects in the code browser, specify `<disable checker="XREFS"/>`.

```
[<disable checker="checker_name"/>
```

`<enable checker="checker_name"/>`
Enable `checker_name`. The checker name is case insensitive. This can be specified multiple times. See also the `<disable checker="checker_name">` element and the `disabledefault="true"` attribute.

```
[<enable checker="checker_name"/>
```

Examples

Analyzing and committing results.

```
```

```
<target name="loadtask" description="Load the covanalyzejava task">
    <taskdef resource="com/coverity/anttask.xml" classpath="${anttask.jar}"/>
</target>

<target name="build.analyzeandcommit" description="Analyze and commit results"
 depends="loadtask">
  <covanalyzeandcommit
    dataport="${env.CIM_COMMIT_PORT}"
    dir="${env.PREVENTINTDIR}"
    host="${env.CIM_SERVER}"
    password="coverity"
    stream="${env.SA_TEST_PROJECT}"
    user="admin"
    version="1.2 rc 7"/>
</target>
```

Enabling all but one of the default checkers for the analysis.

```
<target name="analyzeandcommit.no.forwardnull" depends="loadtask">
  <property environment="env1"/>
  <echo message="Current PATH = ${env1.PATH}"/>
  <covanalyzeandcommit
    dataport="${env.CIM_COMMIT_PORT}"
    dir="${env.SA_INT_DIR}"
    disabledefault="true"
    host="${env.CIM_SERVER}"
    password="coverity"
    stream="${env.SA_TEST_PROJECT}"
    user="admin"
    version="1.2">
    <enable checker="FORWARD_NULL"/>
  </covanalyzeandcommit>
</target>
```

See Also

cov-analyze

covbuild

Name
covbuild Intercept all calls to the compiler invoked by the build system using an Ant task.

Synopsis

Synopsis

```
<covbuild
  dir="int_dir"
  [OPTIONAL_ATTRIBUTES]>
</covbuild>
```

Description

The `covbuild` task calls Ant with a specified build file and target, and it captures any compilations under this call.

Attributes

antargs
: Passes command-line arguments to Ant.

antfile="build.xml"
: Specifies the location of the build file that is called by Ant. The default is the current Ant build file.

binpath="`<install_dir>`/bin"
: Specifies the directory containing `cov-build`. Use this attribute if the Ant task fails to find this command. Without this attribute, the Ant task searches for `cov-build` based on the PATH environment variable and/or the location of `coverity-anttask.jar`.

covbuildargs="options"
: Passes space-delimited options to `cov-build`.

dir="dir"
: Specifies the intermediate directory into which the build is captured.

target="target"
: Specifies a target in the build file (see `antfile`). Defaults to the default target of the specified Ant build file.

Examples

Note that the following are equivalent.

- Ant:

```
<covbuild dir="idir"
  antfile="build0.xml"
  target="build"
/>
```

- Command line:

```
> "cov-build --dir idir ant -f build0.xml build"
```

Additional examples:

```
<target name="build.default" depends="loadtask">
  <property environment="env4"/>
  <echo message="Current PATH = ${env4.PATH}"/>
  <covbuild
    dir="${env.SA_INT_DIR}"
    target="build1"/>
</target>

<target name="build.alternative" depends="loadtask">
  <property environment="env4"/>
  <echo message="Current PATH = ${env4.PATH}"/>
  <covbuild
    dir="${env.SA_INT_DIR}"
    antfile="alternative.xml"
    target="build-alternative"/>
</target>

<target name="build.executable" depends="loadtask">
  <echo message="binpath = ${build.binpath}"/>
  <covbuild
    binpath="${build.binpath}"
    dir="${env.PREVENTINTDIR}"
    target="build1"/>
</target>
```

See Also

cov-build

covanalyzeandcommit

Test Advisor Commands

Name

cov-capture Intercept all calls to the compiler invoked by the build system.

Description

This command, including its command options, is virtually identical to the cov-build command (see differences noted below). The `cov-capture` command is provided for use with Test Advisor so that you can separate your source code builds from your development test code builds if you want to. It can also be used with Coverity Dynamic Analysis as described below.

Differences from `cov-build`:

- The `cov-capture` command writes log data to `<intermediate_directory>/capture-log.txt`.
- The --no-generate-build-id option is a default option for `cov-capture`, not for `cov-build`.
- The `cov-capture` command does not warn when no files are emitted.
- The `cov-capture` command does not save `build-cwd.txt` for use by `cov-analyze --strip-path`.

Using --java-da

Coverity recommends using `cov-capture`, instead of `cov-build`, when running --java-da.

See Also

cov-build

Name

cov-emit-server Start an emit server to collect coverage.

Synopsis

```
cov-emit-server
  [--dir|--idir-library] <directory> [--port <port number>]
  [--interface <ip-to-bind-to>]
  [--gcov-cache-size <size in MB>]
  [--force-start]
```

Description

`cov-emit-server` allows for the collection of coverage from remote machines. `cov-emit-server` can also be used to collect coverage on a single machine. For some tests, this can greatly improve the performance of a coverage collection run. See `cov-build` for information on what coverage tools are supported.

Note

In general, you should not launch `cov-emit-server` directly, `cov-manage-emit` should be used to control the server. For more information, see "Emit Server sub-commands" for `cov-manage-emit`.

`cov-emit-server` creates a log file in `<directory>/cov-emit-server.log`.

Options

--dir `<int_dir>`
: Specifies the intermediate directory in which the emit server will start. Note that `--dir` and `--idir-library` are mutually exclusive, you can only specify one or the other.

--idir-library `<idir-library>`
: Specifies the intermediate directory library in which the emit server will start. Note that `--dir` and `--idir-library` are mutually exclusive, you can only specify one or the other. If `<idir-library>` does not exist, it will be created for you.

--port `<port_number>`
: Specify the port to listen on. Default is 15772.

--interface `<ip-address-to-use>`
: Specify the IP address to listen on. This must be an IPv4 address.

--gcov-cache-size `<size-in-mb>`
: Specify the size of the cache to use for collecting gcov data files. The default size is 500MB.

--force-start
: Force starting the server, even if the pid file exists. This is useful if the previous server did not shut down cleanly.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

Examples

Start an emit server on an intermediate directory:

```
cov-emit-server --dir idir
```

Start an emit server, using a custom port on an external interface:

```
cov-emit-server --dir idir --port 12345 --interface 10.0.0.1
```

Start an emit server on an idir library:

```
cov-emit-server --idir-library my-idir-library
```

See Also

cov-build, cov-capture

cov-manage-emit

Name

cov-emit-server-control Control a running emit server instance

Synopsis

```
cov-emit-server-control
  [--dir <idir>|--idir-library <directory>|--interface <ip-address>]
  [--port <port number>]
  [--start-suite <suitename>|--start-test <testname>|--stop-test <testname>]
  [--max-wait-time <timeout-seconds>] [--status <test-status>]
  [--log <logfile>]
```

Description

`cov-emit-server-control` controls a running instance of `cov-emit-server`. This is used to start and stop tests when using Remote Test Separation.

Options

--dir `<idir>`
: Specifies the intermediate directory of the emit server to control. Exactly one of `--dir`, `--idir-library`, or `--interface` must be specified.

--idir-library `<directory>`
: Specifies the intermediate directory library of the emit server to control. Exactly one of `--dir`, `--idir-library`, or `--interface` must be specified.

--interface `<ip-address>`
: Specifies the IP address on which the emit server is listening. Exactly one of `--dir`, `--idir-library`, or `--interface` must be specified.

--port `<port_number>`
: Specifies the port number on which the emit server is listening. This must be used when `--interface` is given.

--start-suite `<suitename>`
: Start a testsuite with the given suitename. Exactly one of `--start-suite`, `--start-test`, or `--stop-test` must be specified.

--start-test `<testname>`
: Start a test with the given testname. Exactly one of `--start-suite`, `--start-test`, or `--stop-test` must be specified.

--stop-test `<testname>`
: Stop a test with the given testname. Exactly one of `--start-suite`, `--start-test`, or `--stop-test` must be specified.

--max-wait-time `<timeout-seconds>`
: Specifies the timeout to use when communicating with the emit server. If unspecified, a timeout of 30 seconds is used.

--status `<test-status>`
: Specifies the test status to assign to the currently running test. This option should be used with the `--stop-test` option. Acceptable values for test-status are "pass", "fail", or "unknown".

--log `<logfile>`
: Specifies where diagnostic information shall be logged. When logfile is "-", logging is written to stdout. If not specified, no logging is performed.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

Examples

Start a testsuite named "mysuite" on the emit server running on intermediate directory "idir":

```
cov-emit-server-control --dir idir --start-suite mysuite
```

Start a test name "mytest" on the emit server running on IP address 127.0.0.1, port 6405:

```
cov-emit-server-control --interface 127.0.0.1 --port 6405 --start-test mytest
```

See Also

cov-emit-server

Name

cov-extract-scm Extracts the change data for each line from files in the SCM system.

Synopsis

```
cov-extract-scm --scm <scm_type> --output <output_file> --input <input_file>
[--tool <tool_path>] [--project-root <root_path>] [--tool-arg <tool_args>]
[--command-arg <command_arg>] [--log <log_path>] [--ms-delay <int>] [--get-
baseline-code-version][--get-modified-files][--OPTIONS]
```

Description

Queries an SCM system for information of use to Test Advisor, Automatic Owner Assignment and Fast Desktop. `cov-extract-scm` operates in one of two modes: "Annotate" (the default mode) or "Code version".

Annotate mode
: Retrieves the change data for each line of a file from the SCM system. Test Advisor test policy files (see .Creating Test Advisor Policies) might require the age of each line in source code files. See also Automatic Owner Assignment. This is the default mode.

Code version mode
: Retrieves information regarding the code version the user has most recently checked out, updated from, or pulled. The information is either about the code version itself or the unpublished changes since the code version. (See `cov-run-desktop`.)

  This mode is activated by the `--get-baseline-code-version` or `--get-modified-files` options.

Using SCM argument tags

`cov-extract-scm` issues a call to the SCM system to get information about last modified dates for each line in a source file. On most systems, this command is either "blame" or "annotate". For example:

```
accurev annotate foo.c
```

In some cases, SCM systems have additional items that need to be specified in the command to allow Test Advisor to get the appropriate information from the system. The `--tool-arg` and `--command-arg` options allow for this functionality. Furthermore, the `--tool` allows you to specify an SCM tool that is non-standard or provides command output in a format that is not easily parsed by Test Advisor. For example:

```
<tool> <tool-args> <command> <command-args> <coverity-mandated-flags> <target-file>
```

- `<tool>` is from the `--tool` argument. If it is not specified, an appropriate default is used for the specified SCM type.
- `<tool-args>` and `<command-args>` are collections of singular events derived from `--tool-arg` and `--command-arg`, respectively.

- <command> is specific to each source control management system and can not be modified; typically a variation of blame or annotate. If the command is not specified, Test Advisor uses the appropriate command for the specified SCM type.
- <coverity-mandated-flags> are also specific to each source control management system and can not be modified.
- <target-file> is generated from the data passed in the --input option.

The <command> and <coverity-mandated-flags> syntax for each supported SCM is as follows:

- accurev:

  ```
  <tool-args> annotate <command-args> -fudtv <file>
  ```

- clearcase:

  ```
  <tool-args> annotate <command-args> -out - -nheader -f -fmt "revision %X
  \nauthor %u\nusername %Lu\ndate %d\n\t" <file>
  ```

- cvs:

  ```
  log <file>
  ```

  ```
  <tool-args> annotate -F <command-args> <file>
  ```

  Note

  For CVS, <command-args> and <tool-args> are NOT passed to both commands, they are only applied to the annotate command.

- git:

  ```
  <tool-args> blame <command-args> -p <file>
  ```

- hg:

  1. ```
     <tool-args> log <command-args> -f --template \
     ```

     ```
     'author {author|person}\nusername {author|user}\ndate \
     ```

     ```
     {date|hgdate}\nchangeset_long {node}\nchangeset_short {node|short}\n---
     \n'<file>
     ```

  2. ```
     <tool-args> annotate <command-args> -c <file>
     ```

  Any tool-args/command-args are passed to both commands.

- perforce:

  1. ```
     <tool-args> changes <command-args> -t -i <file>
     ```

  2. ```
     <tool-args> annotate <command-args> -q -I <file>#have
     ```

Use `perforce` for versions later than 2010.1.

Any `tool-args`/`command-args` are passed to both commands.

- `perforce2009`:

  1. `<tool-args> changes <command-args> -t -i <file>`

  2. `<tool-args> annotate <command-args> -q -i <file>#have`

  Use `perforce2009` for all perforce versions from 2010.1 and earlier.

  Any `tool-args`/`command-args` are passed to both commands.

- `svn`:

  `<tool-args> blame <command-args> -xml <file>`

- `tfs`:

  You must select the appropriate flag that matches the version of Team Foundation Server that you are using (`tfs2008`, `tfs2010`, `tfs2012`, `tfs2013`, or `tfs2015`). There is no version auto-detection.

  Test Advisor uses a custom application to (`cov-internal-tfs`) to gather information from the Team Foundation Server tools. Because of this, the `<tool-args> and <command-args>` arguments are rejected, and the underlying commands to TFS can not be modified.

Note

The value for `<file>` is almost always the file name and not a file path given in `--input`. Test Advisor changes to the directory where the file resides and issues the command. The exception to this is with `accurev` using the `--project-root` option. In this case, Test Advisor changes to the directory where the file resides but `<file>` is a filepath that is relative to the project root. This is because a valid deployment model for `accurev` is a detached workspace, in which file information must be called with a relative path to the root of where it was checked out.

Additionally, when TFS named items (such as directories, files, and branches) are renamed, some of the historical information (in particular changeset history) is potentially not retrievable for modifications that occurred before the rename. Because of this, `cov-import-scm`/`cov-extract-scm` is unable to properly associate SCM data on a per line basis.

When modifications are performed on a branch and then later merged to a second branch, running `cov-import-scm`/`cov-extract-scm` on the second branch will indicate that all modifications occurred at the date of the merge, rather than the date they actually occurred.

A note on Git superprojects

Git superprojects are unsupported by `cov-extract-scm`'s code version mode (activated by the `--get-baseline-code-version` and `--get-modified-files` options).

You may be able to workaround this issue by creating a script to access Git using submodules, and specify that script with the `--tool` option. This is an advanced usecase, and should only be attempted by experienced users.

Options

--command-arg <command_arg>
(Annotate mode only)

Specifies additional arguments that are passed to the command that retrieves the last modified dates. The arguments are placed after the command and before the target file. This option can be specified multiple times.

--error-threshold <percentage>
Sets the percentage of successful extractions required for `cov-extract-scm` to exit with a success return code (`0`). If the extraction rate is below this threshold, `cov-extract-scm` will print a warning and exit with return code `8`. The default percentage is `80`.

--get-baseline-code-version
(Code version mode only)

This switch indicates that instead of performing its usual function of querying the SCM for "annotate" information, the tool shall write to its `--output` file some information about the code version that the user has most recently checked out, updated from, or pulled.

The output shall be a JSON file using ASCII character encoding and platform-native line endings. It consists of a single JSON object with a single attribute called "date" using `YYYY-MM-DDThh:mmZ` syntax.

Example output file:

```
{
  date: "2013-12-18T15:34Z"
}
```

--get-modified-files
(Code version mode only)

This switch also indicates to suppress normal processing and instead retrieve the set of files with unpublished local changes. These are the files with differences relative to the version of the code indicated by `--get-baseline-code-version`.

The output shall be written to the `--output` file as JSON using ASCII character encoding and "\u" escapes in strings as needed to represent non-ASCII characters. The output is a single JSON object with two attributes, "`modified_files`" and "`untracked_files`". The former are files that the SCM knows about and have been modified from their baseline version. The latter are files that are not checked in to the SCM and are also not excluded by SCM "ignore" filtering (like `.gitignore`). Each is an array of strings representing the file names. File names have their letter case preserved and use the platform native syntax for file names and directory separators. The file names shall be

relative to the repository root, which is assumed to be the current directory unless a different root is specified as a command line option with `--project-root`.

Example output file (using Windows separators):

```
{
  modified_files: [
    "utilities\\cov-format-errors\\cov-format-errors.cpp",
    "Makefile"
  ],
  untracked_files: [
    "analysis\\cov-run-desktop\\some-new-file.cpp",
    "analysis\\cov-run-desktop\\some-new-file.hpp",
    "name with \u1234 non-ASCII character"
  ]
}
```

--input <input_file>
(Annotate mode only)

Specifies the path to a file that contains information about the files that gather last modified dates. The format of this file is the same as the output of the `list-scm-unknown` option of `cov-manage-emit`.

This option should ONLY be used within the workflow described in the "Filtering line queries" section of the *Coverity Test Advisor and Test Prioritization 8.0 User and Administrator Guide*.

--log <log_path>
Specifies the path to a file to which output from the `--tool` executable and other recoverable errors are written.

--ms-delay <int>
Specifies a delay in milliseconds between calls to the underlying SCM. This is useful for preventing a denial of service situation.

--output <output_path>
Specifies the path to a file to which the output data is written to. The format of this output (in Annotate mode) is used as input to the `add-scm-annotations` subcommand for `cov-manage-emit`. See --get-baseline-code-version and --get-modified-files for the format of this output in Code version mode.

--project-root <root_path>
Specifies a path that represents the root of the source control repository.

In Annotate mode, this option is only used when specifying `accurev` as the value to `--scm`. When this is used, all file paths that are used to gather information are interpreted as relative to this project-root path.

In Code version mode, this option allows `cov-extract-scm` to run from a directory other than the root of the source control repository. All filenames returned by `--get-modified-files` are relative to this path.

--scm <scm_type>
Specifies the name of the source control management system. For this option to work, the command requires that source files remain in their usual locations in the checked-out source tree. If the files are copied to a different location after checkout, the SCM query will not work.

The `scm_type` values:

- For Accurev: `accurev`

  (Annotate mode only)

- For ClearCase: `clearcase`

- For CVS: `cvs`

  (Annotate mode only)

- For GIT: `git`

- For Mercurial: `hg`

  (Annotate mode only)

- For Perforce, the properties are: `perforce|perforce2009`

  Use `perforce` for versions later than 2010.1. Use `perforce2009` for all perforce versions from 2010.1 and lower.

- For SVN: `svn`

- For Team Foundation Server: `tfs{2008|2010|2012|2013|2015}`

  For example, for version TFS 2013, use `tfs2013`. This option must match the version of TFS that you are using. These options are Windows only.

  (Annotate mode only)

Note

The following commands or setup utilities must be run before `cov-extract-scm` in order to successfully communicate with :

- `accurev`:

  `login command`

- `perforce`:

  environment variables, particularly `P4PORT`

- `tfs`:

Windows credentials in Credential Manager to access the TFS server

--tool <tool_path>
Specifies the path to an executable that interacts with the source control repository. If the executable name is given, it is assumed that it can be found in the path environment variable. If not provided, the command uses the default tool for the specified `--scm` system.

--tool-arg <tool_args>
Specifies additional arguments that are passed to the SCM tool, specified in the `--tool` option, that gathers the last modified dates. The arguments are placed before the command and after the tool. This option can be specified multiple times.

Shared options

--debug, -g
Turn on basic debugging output.

--ident
Display the version of Coverity Analysis and build number.

--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>
Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

See Also

cov-blame

cov-import-scm

cov-manage-emit

cov-run-desktop

Name

cov-import-scm Collects change data for source files from the SCM.

Synopsis

```
cov-import-scm --scm <scm_type> --dir <intermediate_directory> [--filename-
regex <regex>] [--tool <tool_path>] [--project-root <root_path>] [--tool-
arg <tool_arg>] [--command-arg <command_arg>] [--log <log_path>] [--ms-delay
<int>] [--OPTIONS]
```

Description

The `cov-import-scm` command simplifies the process of retrieving the SCM change data for source files and adding them to the emit directory. This command automates the following command line flow:

1. `cov-manage-emit list-scm-unknown`

2. `cov-extract-scm`

3. `cov-manage-emit add-scm-annotations`

Options

--command-arg <`command_arg`>
: Specifies additional arguments that are passed to the command that gathers the last modified dates. The arguments are placed after the command and before the target file. For usage information, see `cov-extract-scm`.

--dir <`intermediate_directory`>
: Specifies the intermediate directory that is used to store the emit repository.

--error-threshold <percentage>
: Sets a threshold for the percentage of successful extractions (from `cov-extract-scm`) below which this import command (`cov-import-scm`) will display a warning, indicating the need to check for a potential problem. Note, however, that `cov-import-scm` will attempt to add all successful extractions to the emit. The default percentage is `80`.

--filename-regex <`regex`>
: Allows finer control over SCM information gathering. Information is gathered only for filenames that match the regex. Any files that do not match are skipped. This is beneficial when there are specific locations where code is known to exist under SCM control and other locations where it is not (such as system headers).

  For more information, see *Coverity Test Advisor and Test Prioritization 8.0 User and Administrator Guide*.

--log <`log_path`>
: Specifies the path to a file to which executable and other recoverable errors encountered in `cov-extract-scm` are written.

--ms-delay <int>
Specifies a delay in milliseconds between calls to the underlying SCM. This is useful for preventing a denial of service situation.

--project-root <root_path>
Specifies a path that represents the root of the source control repository. This option is only used when specifying accurev as the value to --scm. When this used, all file paths that are used to gather information are interpreted as relative to this project-root path. For usage information, see cov-extract-scm.

--scm <scm_type>
Specifies the name of the source control management system. For this option to work, the command requires that source files remain in their usual locations in the checked-out source tree. If the files are copied to a different location after checkout, the SCM query will not work.

The scm_type values follow:

- For Accurevs: accurev
- For ClearCase: clearcase
- For CVS: cvs
- For GIT: git
- For Mercurial: hg
- For Perforce: perforce|perforce2009

  Use perforce for versions later than 2010.1. Use perforce2009 for all perforce versions from 2010.1 and earlier.
- For SVN: svn
- For Team Foundation Server: tfs{2008|2010|2012|2013|2015}

  For example, for version TFS 2013, use tfs2013. This option must match the version of TFS that you are using. These TFS options are Windows only.

  The <tool-args> and <command-args> arguments are rejected if you choose one of these types. For more information, see cov-extract-scm.

Note

The following commands or setup utilities must be run before cov-import-scm in order to successfully communicate with Test Advisor:

- accurev:

  login command
- For Perforce, the properties are: perforce|perforce2009

Use `perforce` for versions later than 2010.1. Use `perforce2009` for all perforce versions from 2010.1 and earlier.

- `tfs`:

  Windows credentials in Credential Manager to access the TFS server

--tool `<tool_path>`
Specifies the path to an executable that interacts with the source control repository. If the executable name is given it is assumed that it can be found in the path environment variable. if not provided uses the default tool for the specified `--scm` system. For usage information, see `cov-extract-scm`.

--tool-arg `<tool_arg>`
Specifies additional arguments that are passed to the SCM tool specified in the `--tool` option that gathers the last modified dates. The arguments are placed before the command and after the tool. This option can be specified multiple times. For usage information, see `cov-extract-scm`.

Shared options

--debug, -g
Turn on basic debugging output.

--ident
Display the version of Coverity Analysis and build number.

--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>
Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

See Also

cov-extract-scm

cov-manage-emit

Name

cov-manage-history Manages historical data needed by Test Advisor

Synopsis

```
cov-manage-history --dir <intermediate_directory> download | import
[command_options]
```

Description

The `cov-manage-history` command queries a running Coverity Connect instance for the history of functions previously committed to a specified stream. This history is downloaded and stored in a given intermediate directory.

This command replaces the `cov-download-history` command, which is deprecated as of the 7.0 release.

Options

Common options

--dir <intermediate_directory>
: Pathname to the intermediate directory to store the history.

--java
: Deprecated in version 7.0: Specify that the stream contains Java data. Starting in 7.0, the command detects the source code language automatically, so this option is not needed anymore.

Individual commands

download
: Replaces the functionality in `cov-download-history` command. This command has the following options:

  - `--authenticate-ssl`

    This is equivalent to `--on-new-cert distrust`.

  - `--certs <filename>`

    In addition to CA certificates obtained from other trust stores, use the CA certificates in the given `<filename>`. For information on the new SSL certificate management functionality, please see *Coverity Platform 8.0 User and Administrator Guide*

  - `--connect-timeout <n>`

    Sets the timeout for establishing connections to <n> seconds. If a connection to Coverity Connect cannot be established within this time, the transaction is aborted. This timeout cannot be disabled. The default value is 60 seconds.

- `--host <coverityconnect_host>`

  `--port <coverityconnect_port>`

  The hostname and port of the Coverity Connect instance to download history from. `--port` is an optional property. If `--port` is not specified on the command line, the default is 8080 without `--ssl` and 8443 with `--ssl`.

- `--max-retries <n>`

  Sets the number of times to retry failed or aborted requests with Coverity Connect to <n>. Note that this does not include the initial attempt, so a setting of 1 results in at most 2 request attempts. A setting of 0 means to never retry failed requests. The default value is 1.

- `--merge`

  Normally, the download command overwrites any previously downloaded or imported history in the intermediate directory with the newly-downloaded history. Specifying this option causes `cov-manage-history` to instead merge the newly-downloaded history with any existing history. This allows history from multiple streams to be combined for use in analysis.

- `--on-new-cert <trust | distrust>`

  Indicates with `--ssl` whether to trust (with trust-first-time) self-signed certificates, presented by the server, that the application has not seen before.

- `--response-timeout <n>`

  Sets the response timeout to <n> seconds. For every request for data sent to Coverity Connect, if a response is not received within this time, the request is aborted. A setting of 0 disables this timeout. The default value is 300 seconds.

- `--sleep-before-retry <n>`

  Sets the time to sleep before retrying a failed or aborted request with Coverity Connect to <n> seconds. A setting of 0 disables this sleep. The default value is 1 second.

- `--ssl`

  Enables SSL encryption for communication with Coverity Connect.

- `--stream <stream>`

  The name of the stream on the Coverity Connect instance to query.

- `--unstrip-path <unstrip_path>`

  This option is deprecated in 8.0. Use the `--strip-path` option for `cov-analyze` for C/C++, or `cov-analyze-java` for Java.

- `--user <username>`

`--password <password>`

The username and password used to log into the Coverity Connect instance. These will be encrypted if `--ssl` is used.

import
: Responsible for copying the analysis history data from one intermediate directory to another intermediate directory (or the same, if the directory is to be reused). It has the following option:

- `--from-dir <int_dir>`

  The source intermediate directory. This command will copy the analysis history data from the source intermediate directory into the current intermediate directory.

  Note

  You should NOT use the `--from-dir` in your production environment to copy Test Advisor history from an intermediate directory for the first time after a version upgrade (for example, from 7.0.x to 7.5.0).

  Instead, you should commit the "old" intermediate directory to Coverity Connect and then download the Test Advisor history using the `download` option to `cov-manage-history`. For example:

  ```
  cov-manage-history download --host <host_name> --stream <stream_name>    \
    --user <username> --password <password>
  ```

Normally, when the analysis history is generated by `cov-analyze` in an intermediate directory, it can only be committed to an instance of Coverity Connect. If the history is to be re-used within the same intermediate directory (for example, during a subsequent run of analysis on the same intermediate directory), then the import command should be used to make that analysis history available for re-use. To do this, the same intermediate directory should be specified for both the `--dir` and `--from-dir` options.

Shared options

--verbose <0, 1, 2, 3, 4>, -V <0, 1, 2, 3, 4>
: Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.

- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

See Also

cov-capture

cov-extract-scm

cov-import-scm

cov-manage-emit

Name

cov-patch-bullseye Patch a Bullseye small runtime for use with Test Advisor.

Synopsis

`cov-patch-bullseye` --bullseye-dir <`path`> [--output-dir <path>]

Description

The `cov-patch-bullseye` allows you to patch your Bullseye small runtime for Test Advisor coverage. You must build the Bullseye small runtime before you patch it and use it in `cov-build` or `cov-capture`. For usage instructions, see *Coverity Test Advisor and Test Prioritization 8.0 User and Administrator Guide*. in the *Coverity Test Advisor and Test Prioritization 8.0 User and Administrator Guide*.

Options

--bullseye-dir <`path`>
: Path to the Bullseye installation directory.

--output-dir <`path`>
: Path where the modified Bullseye small runtime files will be placed. If this option is not specified, it will default to <`path`>`/coverity`, where <`path`> is the value from `--bullseye-dir`.

Exit codes

Most Coverity Analysis commands can return the following exit codes:

- 0: The command successfully completed the requested task.
- 1: The requested task is complete, but it did not return (or find) any results. Note that some Coverity Analysis commands do not return this error code.
- 2: The command was unable to complete the requested task. This error typically includes an error message and some remediation advice.
- 4: An unexpected error occurred. This error should not occur when the product is used in a supported way. Very likely, the requested task was not completed. This error typically provides some diagnostic and/or debugging output, such as a stack trace.

For exceptions, see cov-commit-defects, cov-analyze, and cov-build.

See Also

cov-build

cov-capture

Coverity Dynamic Analysis Commands

Name

cov-start-da-broker Start the Coverity Dynamic Analysis Broker.

Synopsis

```
cov-start-da-broker --dir <intermediate_directory> --host
<cim_server_host_name> --user <user_name> --password <password> --stream
<dynamic_stream> { --dir <intermediate_directory> } [OPTIONS]
```

Description

The `cov-start-da-broker` command starts the Coverity Dynamic Analysis Broker, a process that receives defect data from Coverity Dynamic Analysis Agents and forwards the data to the Coverity Connect. In addition to the command line, some command options can be set in environment variables, and most can be set via configuration file. See the *Coverity Dynamic Analysis 8.0 Administration Tutorial* for more information.

Options

The list below describes the command line options for `cov-start-da-broker`.

`--broker-port` <port>
: Listen on this port for Agent's connections (default 4422).

`--config-file` <file>, -cf <file>
: Specify the path and filename of the Java properties configuration file. This is where you put your configuration file properties.

`--dataport` <port_number>, `-dp` <port_number>, `--port` <port_number>
: Specify the Coverity Connect server port for source and defect commits (Default: 9090).

`--dir` <intermediate_directory>
: Required option that specifies an intermediate directory that was created through the `cov-build` command. The `cov-build` command captures your source and classes. For more information, see `cov-build` in *Coverity Analysis 8.0 User and Administrator Guide*.

`--help, -h`
: Print a help message and exit.

`--host` <hostname>, `-r` <hostname>
: Specify the host name or IP address of the Coverity Connect server.

`--only-listen`, `-ol`
: [Deprecated] This option is deprecated as of version 7.6.0 and will be removed from a future release.

  Listens for Coverity Dynamic Analysis Agents and send their defects to Coverity Connect, but do not commit the source code to Coverity Connect (Default: False, which means the broker listens for agents and commits source code.)

`--only-source-commit`, `-osc`
[Deprecated] This option is deprecated as of version 7.6.0 and will be removed from a future release.

Commit source code to a source stream, but do not listen for Agents. (Default: False, which means the broker commits source code and listens for Agents.)

`--only-test-connection`, `-n`
Validate the command line and configuration, test the connection to the Coverity Connect server, verify that the source and dynamic streams exist, and exit regardless of other command line options. (Default: False, which means the broker operates normally in a non-test mode.)

`--output-dir` <directory>
Change the output directory from the default of `cda_data`. This directory stores a run directory for each invocation of the Broker.

`--password` <password>, `-pa` <password>
Provide a Coverity Connect password.

`--rundir` <run_directory>
Specify a location for the Coverity Dynamic Analysis Broker run directory. This is where the Broker puts its log files and information files such as `broker.started` and `broker.ok`. If the run directory you specify already exists, the Broker moves the old one to `old_<run_dir>_<number>`, where `<number>` is the first number such that the name isn't taken. This option overrides the default behavior where the Broker creates a run directory with a unique name as a new subdirectory under the output directory.

`--run-prefix` <prefix>, `-rp` <prefix>
Change the prefix of the run directory from the default of "`broker`" to `<prefix>`. The run directory is a sub-directory of the output directory where the Broker stores logs and other information from this run.

`--security-file` <license_path>, `-sf` <license_path>
Select a path to a file that contains a valid Coverity Analysis license that is required to commit source code. This option is unnecessary if the license is properly installed. Not available as an Ant attribute. (Default: `<install_dir_ca>/bin/license.dat`)

`--shutdown-after` <seconds>
Specify the number of seconds to wait before the Broker shuts down when no Agents are connected. This convenience option ensures that unused Broker processes terminate when they are no longer required. Specifying `never` means to never shut down automatically. Default: 600 seconds.

`--stream` <dynamic_stream>
Specify the name of the Coverity Dynamic Analysis stream to commit defects and source code.

`--user` <user_name>, `-u` <user_name>
User name for Coverity Connect server. If no user name is specified, the Broker uses the name of the operating system user that invokes it. (Default: Operating System user name.)

`--version`
Print version information.

Examples

The `cov-start-da-broker` command lines below are entered on a single line without returns.

Example command line (dynamic commit and source commit):

```
cov-start-da-broker --host cim.example.com --dataport 9090 --user jdoe
--password secret --stream my_project --dir my_intermediate_dir
```

Example command line (source commit, no dynamic commit):

```
cov-start-da-broker --host cim.example.com --dataport 9090 --user jdoe
--password secret --stream my_project  --dir my_intermediate_dir --only-source-commit
```

Example command line (dynamic commit, no source commit):

```
cov-start-da-broker --host cim.example.com --dataport 9090 --user jdoe
--password secret --stream my_project --dir my_intermediate_dir --only-listen
```

Name

cov-stop-da-broker Stop the Coverity Dynamic Analysis Broker.

Synopsis

This command allows you to manually shutdown a Coverity Dynamic Analysis Broker process.

`cov-stop-da-broker` [clean | early | dirty] [--broker-port <`port_number`>] [--broker-host <`hostname_or_IP`>]

Description

The `cov-stop-da-broker` command sends a shutdown request to the Coverity Dynamic Analysis Broker (see the *Coverity Dynamic Analysis 8.0 Administration Tutorial* for an explanation of the Broker). If this command is not executed, the Broker automatically terminates 10 minutes after the last Coverity Dynamic Analysis Agent disconnects, although the automatic shutdown period can be changed with the `--shutdown-after` option.

Note

Running `cov-start-da-broker` with the `--only-source-commit` option causes the Broker to exit after finishing its source commit. Running `cov-stop-da-broker` is not necessary in this case.

Options

The list below describes the options for the `cov-start-da-broker` command.

`[clean|dirty|early]`
Specify how the Coverity Dynamic Analysis Broker shuts down. Can be one of three values: `clean`, `dirty` or `early`.

clean
(Default) Tells the Broker to exit once all of the following are true:

- The Broker has finished committing source code to Coverity Connect
- All Coverity Dynamic Analysis Agents have disconnected from the Broker.
- The Broker has finished sending all of the defects that the Coverity Dynamic Analysis Agents reported to Coverity Connect

dirty
Tell the Broker to exit immediately. The Broker terminates the source commit if it has not completed, closes connections to Agents (which will themselves exit if their `failfast` option is true), and drops any defects that it has not sent to Coverity Connect. If the source commit already finished successfully before the dirty shutdown, some defects from this run of the Broker may appear in Coverity Connect, otherwise, no defects from this run of the Broker will appear in Coverity Connect.

early
: Tell the Broker to close its connections to Agents (which will themselves exit if their `failfast` option is true) and then exit when all of the following are true:

  - The Broker has finished committing source code to Coverity Connect.
  - The Broker has finished sending all of the defects that the Agents reported (before having their connections to the Broker forcibly closed) to Coverity Connect.

`--broker-host <host_or_IP>`
: Specify the hostname or IP address on which the Broker is running. Default: localhost

`--broker-port <port-number>`
: Specify the port on which the Broker is listening. Default: 4422

Examples

This example sends a shut down request to the Broker on localhost:4422.

```
cov-stop-da-broker
```

This example sends an early shutdown request to the Broker on host `broker.example.com` on port 1234.

```
cov-stop-da-broker early --brokerhost broker.example.com --brokerport 1234
```

An example using Ant:

```
<cov-stop-da-broker
    shutdowntype="early"
    brokerhost="broker.example.com"
    brokerport="1234"/>
```

Coverity Dynamic Analysis Ant Tasks

Name

cov-dynamic-analyze-java Use Ant task to run a Java program with Coverity Dynamic Analysis Agent enabled.

Description

This task runs a Java program with the Coverity Dynamic Analysis Agent enabled. It functions as a replacement for any existing `java` task. You can replace `<java>` with this task in an existing build script.

This task sets the fork parameter to *`true`*. You cannot run Coverity Dynamic Analysis from within the same virtual machine as the Ant build.

You can disable Coverity Dynamic Analysis's functionality by setting the *`enabled`* parameter to *`false`*. In this mode, the task passes through all functionality to the `java` task, and all parameters that are specific to Coverity Dynamic Analysis are ignored.

The Coverity Dynamic Analysis distribution includes an `antlib.xml` file, so you can enable this task by adding the following line to the Ant build script:

```
<typedef resource="com/coverity/anttask.xml"
  classpath="<install_dir_ca>/library/coverity-anttask.jar"/>
```

This task extends the `java` ant task, so it supports all parameters that the `java` task supports.

Attributes

Below are the Coverity Dynamic Analysis Ant attributes for `cov-dynamic-analyze-java` and `cov-dynamic-analyze-junit`. Default values are in parenthesis.

`detect_deadlocks="<boolean>"`
: Detect deadlocks. (True)

`agentoptions="<boolean>"`
: Specify a list of Coverity Dynamic Analysis Agent options as you would pass on the command line in the form of `option=value,option=value...` This attribute allows you to specify Agent options for which there is no Ant attribute. Do not specify an Agent option both in this string and via an Ant attribute because doing so has undefined consequences. (Not specified.)

`detect_races="<boolean>"`
: Detect race conditions. (True.)

`detect_resource_leaks="<boolean>"`
: Detect resource leaks. (True.)

`broker_host="<host_or_IP>"`
: Specify the hostname or IP address on which you ran the Broker. (localhost)

`broker_port="<port_number>"`
: Specify the port on which the Broker is listening. This is set with the `broker-port` option to `cov-start-da-broker`. If you intend to run more than one Broker instance simultaneously on the same machine, it is a good practice to use non-default ports to avoid collisions. (4422)

`enabled=<boolean>`
(No agent option.) Enable Coverity Dynamic Analysis. If set to False, then a `cov-dynamic-analyze-java` task behaves like a `java` task, and a `cov-dynamic-analyze-junit` task behaves like a `javaunit` task. (True.)

`exclude_instrumentation="<colon_separated_list_of_prefixes>"`
Exclude classes from being watched by Coverity Dynamic Analysis to speed up Coverity Dynamic Analysis. However, Coverity Dynamic Analysis does not detect defects in excluded code nor as a result of actions performed in excluded code (such as field access or lock acquisitions).

This option consists of a colon-separated list of prefixes of the fully qualified names to exclude. For example:

"`exclude-instrumentation=A.B`" excludes any class whose name starts with "`A.B`", such as "`A.B`", "`A.B.c`", or "`A.Bc`".

"`exclude-instrumentation=A.B.`" (with a period) excludes "`A.B.c`", but not "`A.B`" nor "`A.Bc`".

(The default is to exclude nothing.)

`failfast="<boolean>"`
If True, the Coverity Dynamic Analysis Agent exits the program it is watching if the Coverity Dynamic Analysis Agent loses its connection to the Broker or has other problems. If False, the Coverity Dynamic Analysis Agent quietly allows the program to continue running, even if Coverity Dynamic Analysis Agent cannot run properly. (False.)

`instrument_only="<colon_separated_list_of_prefixes>"`
Specify a colon-separated list of classes watched by Coverity Dynamic Analysis. Coverity Dynamic Analysis excludes all other classes (same as if they were all specified as options to `exclude-instrumentation`). As with `exclude-instrumentation` above, using this option might speed up Coverity Dynamic Analysis, but at the cost of missing defects. This option consists of a colon-separated list of prefixes of fully qualified names to include. For example:

"`instrument-only=A.B`" includes any class whose name starts with "`A.B`", such as "`A.B`", "`A.B.c`", or "`A.Bc`".

"`instrument-only=A.B.`" includes "`A.B.c`", but not "`A.B`" nor "`A.Bc`".

(The default is to include everything.)

`instrument_arrays="<boolean>"`
Watch reads and writes into arrays to report race conditions. (False.)

`instrument_collections="<boolean>"`
Detect race conditions on collections. For example suppose `map` is a `java.util.Map` and one thread executes `map.put("key", "value")` without holding any locks. If Coverity Dynamic Analysis sees another thread access this map, it reports a `RACE_CONDITION`. (True.)

`override_security_manager="<boolean>"`
Install a permissive security manager (`java.lang.SecurityManager`) that allows all operations except the installation of other security managers. This option exists because a restrictive security

manager causes Coverity Dynamic Analysis to fail. Setting this option to `true` might allow Coverity Dynamic Analysis of your program to proceed. If this option is `false`, running Coverity Dynamic Analysis on your application might require adjusting your security policy or excluding classes that run within the restrictive security manager (see the `exclude-instrumentation` and `instrument-only` options). (False.)

`repeat_connect="<non-negative_number>"`
: If this option is set to a number greater than zero, and the initial attempt to connect to the Broker fails, then the Agent tries to reconnect to the Broker that number of times, with a one second pause between attempts, before giving up. For best results, set this option to something greater than zero when starting both the Broker and Agents from a script or build file. (0 when running the Agent from the command line and 20 when running it through the `cov-dynamic-analyze-java` or `cov-dynamic-analyze-junit` Ant tasks.)

Nested elements

There are no additional nested elements beyond what the `java` or `junit` tasks support.

Examples

The following example runs the program `test.Main` with Dynamic Analyzer instrumentation enabled:

```
<cov-dynamic-analyze-java classname="test.Main" detect_deadlocks="false">;
  <arg value="-h">
  <classpath>
    <pathelement location="dist/test.jar"/>;
    <pathelement path="${java.class.path}"/>
  </classpath>
</cov-dynamic-analyze-java>
```

Name

cov-dynamic-analyze-junit Runs tests from the JUnit testing framework with Coverity Dynamic Analysis enabled.

Description

This task runs tests from the JUnit testing framework with Coverity Dynamic Analysis enabled. It functions as a replacement for any existing JUnit task. You can replace `<junit>` with `<cov-dynamic-analyze-junit>` in an existing build script.

This task sets the fork parameter to *`true`*. You cannot run Coverity Dynamic Analysis from within the same virtual machine as the Ant build. However, all the tests can be run within a single forked VM by setting the *`forkmode`* parameter of the `junit` task to *`once`*.

Disable Coverity Dynamic Analysis's functionality by setting the *`enabled`* parameter to *`false`*. In this mode, the task passes through all functionality to the `junit` task and all Coverity Dynamic Analysis-specific parameters are ignored.

The Coverity Dynamic Analysis distribution includes an `antlib.xml` file, you can enable this task by adding the following line to the Ant build script:

```
<typedef resource="com/coverity/anttask.xml"
classpath="<install_dir_ca>/library/coverity-anttask.jar"/>
```

This task extends the `junit` ant task. It supports all parameters that by the `junit` task supports. For the complete list of supported parameters, see the Ant `junit` task documentation.

Attributes

For the available `cov-dynamic-analyze-junit` attributes, see cov-dynamic-analyze-java(*Coverity 8.0 Command and Ant Task Reference*).

Nested elements

There are no additional nested elements beyond what the `java` or `junit` tasks support.

Examples

The following example runs the test that is defined in `my.test.TestCase` with Coverity Dynamic Analysis instrumentation enabled:

```
<cov-dynamic-analyze-junit>
  <test name="my.test.TestCase"/>
  </cov-dynamic-analyze-junit>
```

The following example runs the test that is defined in `my.test.TestCase` with Coverity Dynamic Analysis instrumentation enabled. At the end of the test, a one-line summary prints. A detailed report of the test is in `TEST-my.test.TestCase.txt`. The build process stops if the test fails.

```
<cov-dynamic-analyze-junit printsummary="yes" haltonfailure="yes">
  <formatter type="plain"/>
```

```
  <test name="my.test.TestCase"/>
</cov-dynamic-analyze-junit>
```

The following example runs `my.test.TestCase` in the same VM with Coverity Dynamic Analysis instrumentation disabled.

```
<cov-dynamic-analyze-junit enabled="no">
  <test name="my.test.TestCase"/>
</cov-dynamic-analyze-junit>
```

Name

cov-start-da-broker Start the Coverity Dynamic Analysis Broker using an Ant task.

Synopsis

```
<cov-start-da-broker ATTRIBUTES/>
```

Description

The `cov-start-da-broker` Ant task starts the Coverity Dynamic Analysis Broker, which receives defects from Coverity Dynamic Analysis instances and sends them to Coverity Connect. See *Coverity Dynamic Analysis 8.0 Administration Tutorial* for details on what the Broker does and how it works.

Attributes

The list below describes the Ant attributes for `cov-start-da-broker`.

`brokerport="<port>"`
: Listen on this port for Agent's connections (default 4422).

`configfile="<file>"`
: Specify the path and filename of the Java properties configuration file. This is where you put your configuration file properties.

`dataport="<port_number>"`
: Specify the Coverity Connect server port for source and defect commits (Default: 9090).

`dir="<intermediate_directory>"`
: Required option that specifies an intermediate directory that was created through the `cov-build` command. The `cov-build` command captures your source and classes. For more information, see `cov-build` in *Coverity Analysis 8.0 User and Administrator Guide*.

`host="<hostname>"`
: Specify the host name or IP address of the Coverity Connect server.

`onlylisten="<boolean>"`
: [Deprecated] This option is deprecated as of version 7.6.0 and will be removed from a future release.

  Listens for Coverity Dynamic Analysis Agents and send their defects to Coverity Connect, but do not commit the source code to Coverity Connect (Default: False, which means the broker listens for agents and commits source code.)

`onlysourcecommit="<boolean>"`
: [Deprecated] This option is deprecated as of version 7.6.0 and will be removed from a future release.

  Commit source code to a source stream, but do not listen for Agents. (Default: False, which means the broker commits source code and listens for Agents.)

`onlytestconnection="<boolean>"`
: Validate the command line and configuration, test the connection to the Coverity Connect server, verify that the source and dynamic streams exist, and exit regardless of other command line options. (Default: False, which means the broker operates normally in a non-test mode.)

`outputdir="<directory>"`
Change the output directory from the default of `cda_data`. This directory stores a run directory for each invocation of the Broker.

`password="<password>"`
Provide a Coverity Connect password.

`rundir="<run_directory>"`
Specify a location for the Coverity Dynamic Analysis Broker run directory. This is where the Broker puts its log files and information files such as `broker.started` and `broker.ok`. If the run directory you specify already exists, the Broker moves the old one to `old_<run_dir>_<number>`, where `<number>` is the first number such that the name isn't taken. This option overrides the default behavior where the Broker creates a run directory with a unique name as a new subdirectory under the output directory.

`runprefix="<prefix>"`
Change the prefix of the run directory from the default of "`broker`" to `<prefix>`. The run directory is a sub-directory of the output directory where the Broker stores logs and other information from this run.

`shutdownafter="<seconds>"`
Specify the number of seconds to wait before the Broker shuts down when no Agents are connected. This convenience option ensures that unused Broker processes terminate when they are no longer required. Specifying `never` means to never shut down automatically. Default: 600 seconds.

`stream="<dynamic_stream_name>"`
Specify the name of the Coverity Dynamic Analysis stream to commit defects and source code.

`user="<user_name>"`
User name for Coverity Connect server. If no user name is specified, the Broker uses the name of the operating system user that invokes it. (Default: Operating System user name.)

Examples

- Commits the source code to Coverity Connect. This target uses `cov-start-da-broker` with `onlysourcecommit="true"`. Splitting the source commit from the dynamic defect commit – especially in Ant build files– is a good practice because it ensures that the whole target will fail if the source commit fails.

```
<property name="demos.idir" location="example-idir"/>

<target name="source.commit" depends="build-project">
  <cov-start-da-broker
    failonerror="true"
    configFile="${config.file}"
    dir="${demos.idir}"
    onlySourceCommit="true"
  />
</target>
```

- Starts the Broker in the background. This target uses `cov-start-da-broker` with `onlylisten="true"` to listen for Coverity Dynamic Analysis Agent connections and to stream their defects to Coverity Connect.

```
<target name="broker.listen">
  <cov-start-da-broker
    failonerror="true"
    configfile="${config.file}"
    onlylisten="true"
   />
</target>
```

Sample configuration file:

```
host=cim.example.com
dataport=9090
user=jdoe
password=secret
stream=Example-dynamic
```

Name

cov-stop-da-broker Stop the Coverity Dynamic Analysis Broker using an Ant task.

Synopsis

```
<cov-stop-da-broker ATTRIBUTES/>
```

Description

The `cov-stop-da-broker` Ant task stops the Coverity Dynamic Analysis Broker, which receives defects from Coverity Dynamic Analysis instances and sends them to Coverity Connect.

Attributes

The list below describes the attributes for the `cov-start-da-broker` Ant task.

`shutdowntype="<broker-shutdown-type>"`
: Specify how the Coverity Dynamic Analysis Broker shuts down. Can be one of three values: `clean`, `dirty` or `early`.

  clean
  : (Default) Tells the Broker to exit once all of the following are true:

    - The Broker has finished committing source code to Coverity Connect
    - All Coverity Dynamic Analysis Agents have disconnected from the Broker.
    - The Broker has finished sending all of the defects that the Coverity Dynamic Analysis Agents reported to Coverity Connect

  dirty
  : Tell the Broker to exit immediately. The Broker terminates the source commit if it has not completed, closes connections to Agents (which will themselves exit if their `failfast` option is true), and drops any defects that it has not sent to Coverity Connect. If the source commit already finished successfully before the dirty shutdown, some defects from this run of the Broker may appear in Coverity Connect, otherwise, no defects from this run of the Broker will appear in Coverity Connect.

  early
  : Tell the Broker to close its connections to Agents (which will themselves exit if their `failfast` option is true) and then exit when all of the following are true:

    - The Broker has finished committing source code to Coverity Connect.
    - The Broker has finished sending all of the defects that the Agents reported (before having their connections to the Broker forcibly closed) to Coverity Connect.

`brokerhost="<host_or_IP>"`
: Specify the hostname or IP address on which the Broker is running. Default: localhost

`brokerport="<port_number>"`
Specify the port on which the Broker is listening. Default: 4422

Examples

The example below shows a clean shutdown of the Broker.

```
<target name="broker.shutdown">
  <cov-stop-da-broker shutdowntype="clean"/>
</target>
```

Coverity Connect Commands

Name

cov-admin-db Administer Coverity Connect.

Synopsis

```
cov-admin-db {backup <archive_file> [--force | --no-overwrite]| optimize |
reset-admin-password | restore <archive_file> [--force | --no-overwrite ] [-j
<number_of_processors>] | upgrade-schema} [--debug]
```

Description

The `cov-admin-db` command performs various operations on Coverity Connect. The database options only work on an embedded database. This command also generates a log file, `<install_dir_cc>/logs/cov-admin-db.log`.

In order to ensure that the upgrade process completes successfully for large databases, it might be necessary to increase the Java heap size allocated to the installer. Use the following syntax to set the `INSTALL4J_ADD_VM_PARAMS` environment variable:

```
INSTALL4J_ADD_VM_PARAMS=-Xmx<mem>
```

Where `<mem>` is the heap size. The default is 768MB (`<mem>` = 768m). If the heap size is too large, the installer will not start. The heap size should be set the value to 1/2 total available RAM. For example, `-Xmx8g`.

Options

`backup <archive_file>`
: Back up the database to an archive file. This option only works with the embedded database.

  Note

  All commits started prior to backup invocation will complete before the backup starts.

  Any commit started while backup is running will be put on hold until the backup process is complete.

--force
: Can be optionally used with either the `backup` or `restore` sub-commands. It suppresses user inputs during the backup and restore process to help automate the procedure.

`reset-admin-password`
: Change password for the `admin` account to that specified at the prompt.

`optimize`
: Improve database use of indexes and statistics. This option only works with the embedded database.

  Do not use this option with Coverity Integrity Manager versions 5.0-5.2. This option can be used without putting Coverity Connect in `Maintenance` mode.

Note

To reclaim the maximum amount of space, you can back up and restore the embedded database. For information about this process, see "Administering the Coverity Connect database" in the *Coverity Platform 8.0 User and Administrator Guide*.

`restore <archive_file>`
Restore data to the embedded database, using the specified archive file. For an archive file that was generated from a backup of a previous version of Coverity Connect, this option restores the archived data and upgrades the schema to make the schema compatible with the current version of Coverity Connect. The archive file is not modified during this process.

Before you use this option, run `cov-im-ctl maintenance`.

Use caution when restoring a database with this option because it deletes data from the current database. This option only works with the embedded database.

-j `<number_of_processors>`
Used with the `restore` subcommand to control the level of parallelism and reduce the amount of time to perform the restore. The `<number_of_processors>` attribute sets the number of core processors available on your system. The default is 1.

`upgrade-schema`
Upgrade an archived schema from a previous version of Coverity Connect to make it compatible with the current version of Coverity Connect. The `upgrade-schema` option supports an external PostgreSQL database or the embedded database. This option is most useful when it is necessary to update a schema for an external database that was backed up prior to a Coverity Connect upgrade.

Use caution when upgrading a schema with this option because it is an irreversible process. Back up the current database before using this option, if possible.

--debug
Display debug output only if this option is present on the command line.

--no-overwrite
This option indicates that the following will not be overwritten:

- An existing backup file with the backup command
- The contents of a database with the restore command

If you do not specify this option, you must manually answer the questions during the backup/restore process. All questions are skipped if `--force` is present.

Examples

Change the administrator password:

```
> cov-admin-db reset-admin-password
```

Name

cov-get-certs Create a file of trusted self-signed certificates.

Synopsis

```
cov-get-certs --host <host> --port <port> --certs <certs_file>
```

Description

In situations where Coverity Connect is a client of other services (email, Bugzilla, JIRA, LDAP) and one or more of those services uses a self-signed certificate, `cov-get-certs` is used to transfer that server's self-signed certificate to Coverity Connect's trust store of CA root certificates, thus enabling Coverity Connect to connect using SSL to the service.

Before the Coverity 8.0 release, `cov-get-certs` was needed for all Java applications. Now it is needed only for Coverity Connect

If you want to edit the certificate file, use your JRE's `keytool` command. The password for the certificates file is `changeit`.

Options

--host `<server-host>`
: The Coverity Connect server hostname.

--port `<server-port>`
: The Coverity Connect server HTTPS connection port. The default is `8443`.

--certs `<cert_file>`
: The name of the certificate file. Because this defaults to Coverity Connect's Java trust store (at `<Coverity Connect install directory>/jre/lib/security/cacerts`), you don't normally need to use this option.

Example

```
cov-get-certs --host example.com --port 8443
```

Name

cov-im-ctl Start or stop Coverity Connect.

Synopsis

```
cov-im-ctl {maintenance | start | status | stop} [-w <seconds>]
```

Description

The `cov-im-ctl` command performs various operations, including starting or stopping Coverity Connect.

On Windows systems, when Coverity Connect is installed as a service, the `cov-im-ctl.exe` program is often unnecessary because Coverity Connect starts and stops automatically when the system boots up or shuts down. When Coverity Connect is installed as a service, any administrator can use this command.

When Coverity Connect is not installed as a service, only the user who installed Coverity Connect is able to use this program to start or stop it.

Options

maintenance
: Place the embedded database in maintenance mode. Use this option, for example, before you restore a database from backup.

start
: Start Coverity Connect.

status
: Provide status information about Coverity Connect.

stop
: Stop Coverity Connect.

-w `<seconds>`
: Wait at least the specified number of seconds for a response to this request.

Examples

Check to see if Coverity Connect is running:

```
> cov-im-ctl status
```

Stop Coverity Connect:

```
> cov-im-ctl stop
```

Start Coverity Connect:

```
> cov-im-ctl start
```

Name

cov-import-cert Import certificate for SSL communication for coordinator/subscribers.

Synopsis

```
cov-import-cert <cert_file> <truststore_file>
```

Description

`cov-import-certs` allows you add the certificate into the user's own truststore, to allow authentications between a coordinator and a subscriber. If the truststore does not exist, one will be created. For more information, see *Configuring Coverity Connect enterprise clusters* in the *Coverity Platform 8.0 User and Administrator Guide*.

Options

`<cert_file>`
: The name of the certificate file.

`<truststore_file>`
: The truststore file shared for SSL communication. If the truststore does not exist, one will be created.

Name

cov-manage-im Manage and query defects, projects, and streams in Coverity Connect.

Synopsis

```
cov-manage-im --mode defects [MODE OPTIONS][CONNECTION OPTIONS][SHARED
OPTIONS]

cov-manage-im --mode projects [MODE OPTIONS][CONNECTION OPTIONS][SHARED
OPTIONS]

cov-manage-im --mode streams [MODE OPTIONS][CONNECTION OPTIONS][SHARED
OPTIONS]

cov-manage-im --mode triage [MODE OPTIONS][CONNECTION OPTIONS][SHARED
OPTIONS]

cov-manage-im --mode motd [MODE OPTIONS][CONNECTION OPTIONS][SHARED OPTIONS]

cov-manage-im --mode commit [MODE OPTIONS][CONNECTION OPTIONS][SHARED
OPTIONS]

cov-manage-im --mode notification [MODE OPTIONS][CONNECTION OPTIONS][SHARED
OPTIONS]

cov-manage-im --mode auth-key [MODE OPTIONS][CONNECTION OPTIONS][SHARED
OPTIONS]
```

Description

The `cov-manage-im` command modifies and queries information for defects, projects, and streams in Coverity Connect. This command also outputs logging information to `<install_dir_cc>/logs/cim.log`.

This command has the following modes of operation:

- Defects mode
- Projects mode
- Streams mode
- Triage mode
- MOTD mode
- Commit mode
- Notification mode

- Authentication key mode

The `cov-manage-im` command can operate on (update or delete) the set of objects that were matched by a query within a single command line.

Each `cov-manage-im` mode accepts CONNECTION options that allow you specify connection settings such as host name, port number, and so forth, on the command line.

Alternatively, you can use the `coverity_config.xml` file, which is a configuration file that you can edit to store connection options for `cov-manage-im`. If you run the `cov-manage-im` command from a Coverity Analysis or Coverity Analysis package, you can create a default version of this file at `<install_dir_sa>/config/coverity_config.xml` by running the `cov-configure` command. After you run the `cov-configure` command, you must add the elements as in the example that follows, if you want to include connection options in the configuration file instead of on the command line. If you run the `cov-manage-im` command from a Coverity Connect package, you must manually create the default `coverity_config.xml` file, and move it to `<install_dir_cc>/config/coverity_config.xml`.

The following example element in the `coverity_config.xml` file defines connection options for Coverity Connect:

```
<!DOCTYPE coverity SYSTEM "coverity_config.dtd">
<coverity>
  <config>
    <cim>
      <host>cim.company.com</host>
      <port>8443</port>
      <!-- HTTPS port -->

      <client_security>
         <user>test</user>
         <password>secret</password>
         <ssl>yes</ssl>
         <certs>/pathto/.certs</certs>
      </client_security>
    </cim>
  </config>
</coverity>
```

Note

Glob arguments are patterns used for filter expressions. In glob patterns, * matches zero or more characters, and ? matches exactly one character.

Defects mode

Query and update defects in Coverity Connect.

Synopsis

```
--mode defects --show <SCOPE> [<FILTER>] [<OUTPUT>] [<OTHER>]
```

```
--mode defects --update <SCOPE> [<FILTER>] <SET> [<OTHER>]
```

Defects mode options

In general, you can specify options in any order. The exception is when you add more than one project or stream within a single command. In this case, you must specify the options for the properties of each new stream or project at the same time.

The <OTHER> option listed in the synopsis refers to sets of command options that are common to all modes. The options are:

- Common OUTPUT options
- CONNECTION options
- Shared options

OPERATION options
: Exactly one OPERATION option is required:

  `--show`
  : Output a comma separated value (CSV) list of defects from the specified scope that matches the filter criteria. Use the `--fields` option to control the display of the defect fields and their order.

  `--update`
  : Update attributes of defects from the specified scope that match the filter criteria. At least one SET option is required. FILTER options are not required.

SCOPE
: Define the project or the set of streams to show or update. All defect operations require a scope definition.

  When the same defect occurs in multiple streams, and a project scope is specified, the defects are represented as a `merged defect` for display and filtering purposes. For example, if the same defect occurs in two separate streams, and the action attribute values are different, Coverity Connect calculates a merged action. The merged action is then displayed for the merged defect. FILTER options match against the merged defect.

  The SCOPE options are:

  --project <name>
  : Specify the name of a project that contains the defects that you want to show or update. You can only scope for one project.

  --stream <name>
  : Specify the name of a stream, or streams.

FILTER
: FILTER options focus the set of defects in the given scope that will be operated on. You can specify multiple instances of each filter, with the exception of the `--file` and `--function` filters, which you can only specify once.

If you specify multiple instances of the same filter, the values are effectively ORed together. Different filter options are effectively ANDed together. For example, `--action a --action b --severity c --severity d` matches defects of the criteria `(action = a OR b) AND (severity = c OR d)`.

The FILTER options are:

--cid <cid-set>
: Operate on a single CID or set of CIDs. <cid-set> can be:

  - A single CID. For example, `--cid 10118`.
  - A range of CIDs. Denote range with a hyphen (-). For example, `--cid 10203-10209`.
  - A comma-separated list of single CIDs and ranges of CIDs. For example, `--cid 10118,10119,10203-20109,10388`.

--action <action>
: Operate on defects whose action attribute value exactly matches the <action> string. Valid strings match the list of Action values on the Coverity Connect *Attribute Details* screen. The standard (unedited) values in Coverity Connect are:

  - `Undecided`
  - `Fix Required`
  - `Fix Submitted`
  - `Modeling Required`
  - `Ignore`

  Use `Various` for <action> to select merged defects that have different action attribute values.

--classification|--class <class>
: Operate on defects whose classification attribute value exactly matches the <class> string.

  <class> must be one of the following:

  - `Unclassified`
  - `Pending`
  - `False Positive`
  - `Intentional`
  - `Bug`
  - `Various`

  Use `Various` for <classification> to select merged defects that have different classification attribute values.

For Test Advisor policy violations, the classification must be one of the following:

- `Unclassified`
- `Pending`
- `Untested`
- `No Test Needed`
- `Tested Elsewhere`
- `Various`

--severity `<severity>`
Operate on defects whose severity attribute value exactly matches the `<severity>` string. Valid strings match the list of severity levels on the Coverity Connect *Attribute Details* screen. The standard (unedited) values in Coverity Connect are:

- `Unspecified`
- `Major`
- `Moderate`
- `Minor`

Use `Various` for `<severity>` to select merged defects that have different severity attribute values.

--status `<status>`
Operate on defects whose status attribute value exactly matches the `<status>` string.

`<status>` must be one of the following:

- `New`
- `Triaged`
- `Dismissed`
- `Fixed`

--component `<comp_name>`
Operate on defects found by a specified component name. The name of the component must match the `<comp_name>` string.

--component-not `<comp_name>`
Causes defects found by a specified component name to be excluded from the result. The name of the component must match the `<comp_name>` string. You can specify multiple component names.

--checker <checker>
Operate on defects found by a specific checker. The name of the checker must match the <checker> string. For example:

```
--checker FORWARD_NULL
```

--external-reference|--ext-ref <ext-reference>
Operate on defects whose external reference exactly matches the <ext-reference> string.

--language|--lang< lang>
Operate on a stream, or streams, based on language.<lang> can be one of the following:

- C/C++, cpp
- Java, java
- C#, cs
- dynamic_java
- Mixed, mixed
- Other, other

--mergekey <mergekey>
Operate on defects whose mergekey exactly matches the <mergekey> string. You may specify multiple mergekeys in a comma-separated list.

--owner <owner>
Operate on defects whose owner exactly matches the <owner> string.

Use Unassigned for <owner> to update defects that are not yet assigned.

--file <file>
Operate on defects whose file matches the <file> glob pattern. <file> refers to the terminal part of a filename, not a full path. You can specify this option only once.

The glob <file> <pattern> refers to the entire pathname. Pathnames are separated by a slash on all platforms. For example, `--file *.java` will match all Java files, and `--file Win.java` will match only `Win.java` at the root of the source tree. But `--file */Win.java` will match all files named `Win.java` in all directories.

--function <function>
Operate on defects whose function matches the <function> glob pattern. You can specify this option only once.

--legacy <status>
Operate on defects whose legacy status matches the <status> string. Legacy status can be any of the following values:

- True

- `False`
- `Various`

--newest [snapshotId]
Operate on defects which occur only in the project's most recent snapshot.

An optional parameter, `[snapshotId]`, will compare the newest snapshot with the older specified snapshot. The snapshot ID number must match the number in `[snapshotId]` exactly.

OUTPUT options
OUTPUT options are not required and are only valid with the `--show` option.

--newest [snapshotId]
Outputs those defects which occur only in the project's most recent snapshot.

An optional parameter, `[snapshotId]`, will compare the newest snapshot with the older specified snapshot. The snapshot ID number must match the number in `[snapshotId]` exactly.

--output fields
Display the list of valid field names for this mode. These field names can then be used with the `--fields` option.

--page
Sets the number of defects that are pulled per batch. The default number of defects is 100. If you set the `--page` option to 1000 then there will be fewer queries and performance can improve. For example, by setting the `--page` option to 1000 you will get 20 queries rather than 200, so you will get 10 times fewer.

Example:

```
cov-manage-im --host localhost --port 8080 --user admin --password coverity --
mode defects --show --project foo --page 1000
```

Output options
You can also specify Output options that are common to all modes.

SET options
SET options update defect attributes of the selected defects defined with the FILTER options. You can use only one `--set` option for each defect attribute, such as action, severity, classification, and so forth. However, you can specify `--set` options for different defect attributes on the same command line.

--set action:`<action>`
Set the action defect attribute for defects that match the filter criteria to `<action>`. The action attribute used for `<action>` must already exist in the Coverity Connect database.

--set {classification|class}:`<class>`
Set the classification attribute of defects that match the filter criteria to `<class>`.

The `<class>` string must be one of the following:

- `Unclassified`
- `Pending`
- `False Positive`
- `Intentional`
- `Bug`

For Test Advisor policy violations, the classification must be one of the following:

- `Unclassified`
- `Pending`
- `Untested`
- `No Test Needed`
- `Tested Elsewhere`
- `Various`

--set severity:`<severity>`
Set the severity defect attribute of defects that match the filter criteria to `<severity>`. The severity attribute used for `<severity>` must already exist in the Coverity Connect database.

--set owner:`<owner>`
Set the owner of defects that match the filter criteria to `<owner>`. The owner attribute used for `<owner>` must already exist in the Coverity Connect database.

--set comment:`<comment>`
Add a comment to the defects that match the filter criteria.

--set {ext-ref|external-reference}:`<reference>`
Add an external reference to the defects that match the filter criteria.

--set legacy:`<status>`
Set the legacy status for the defect. The acceptable values are:

- `True`
- `False`

Defects mode examples

The first example shows the four connection options that must contain values either on the command line, or in the XML configuration file (host, port, user, password). These connection options are intentionally dropped from most of the subsequent examples to reduce the length of the command lines. When the connection options are not specified, assume that the values are retrieved from the default XML configuration file.

Show examples

Show all (merged) defects in project X.

```
> cov-manage-im --host cim.company.com --port 8080 --user test \
    --password secret --mode defects --show --project X
```

Show all open defects in project X

```
> cov-manage-im ---mode defects --show --project X \
    --status New --status Triaged
```

Show all defects with the specified mergekey in the stream named Y.

```
> cov-manage-im --mode defects --show --stream Y --mergekey
 46941efa13559f40754b0d90dd99f2d2
```

List the defect fields that can be passed to `--fields` in defects mode.

```
> cov-manage-im --mode defects --show --project P --output fields
```

Control what defect fields are shown by specifying some of these fields with the `--fields` option.

```
> cov-manage-im --mode defects --show --project P \
    --fields cid,action,severity
```

Show particular (merged) defects in project X, filtering with different CID specifiers.

```
> cov-manage-im --mode defects --show --project X --cid 123
    > cov-manage-im --mode defects --show --project X --cid 123,456
    > cov-manage-im --mode defects --show --project X --cid 1-4
    > cov-manage-im --mode defects --show --project X --cid 1-4,18,25-30
```

Show all Triaged defects in stream Y that are classified as Bugs.

```
> cov-manage-im --mode defects --show --stream Y \
    --status Triaged --class Bug
```

Show all open defects in stream Y with no owner.

```
> cov-manage-im --mode defects --show --stream Y \
    --status New --status Triaged --owner Unassigned
```

Show all open defects in "client" source files.

```
> cov-manage-im --mode defects --show --stream Y \
                --status New --status Triaged --file "client-*.c"
```

Show all open defects listed by CID in all components.

```
cov-manage-im --mode defects --show --project project1 \
    --fields cid,component
```

Update examples

Assign all unassigned defects in streams X and Y to `jdoe`

```
> cov-manage-im --mode defects --update --stream X \
    --stream Y --owner Unassigned --set owner:jdoe
```

Set the classification of CID 10002 in project `P` to False Positive.

```
> cov-manage-im --mode defects --update --project P \
    --cid 10002 --set "class:False Positive"
```

Set all the attributes of CID 10002 in project `P`.

```
> cov-manage-im --mode defects --update --project P \
    --cid 10002 --set "action:Fix Required" --set class:Bug \
    --set severity:Major --set owner:jdoe \
    --set "comment:This appears to be real." \
    --set ext-ref:yyy
```

Set the classification of defects with the specified mergekey in stream `Y` to False Positive.

```
> cov-manage-im --mode defects --update --stream Y \
    --mergekey 46941efa13559f40754b0d90dd99f2d2 --set "class:False Positive"
```

Mark all defects in `MyStream` which were introduced in the most recent snapshot as Legacy=true.

```
> cov-manage-im --mode defects --stream MyStream --update --newest --set legacy:true
```

Mark all defects in `MyStream` which were introduced after snapshot 10006 as Legacy=true.

```
> cov-manage-im --mode defects --stream MyStream --update --newest 10006 --set
 legacy:true
```

Projects mode

Query, add, delete, and update projects in Coverity Connect.

Synopsis

```
--mode projects --show [<FILTER>] [<OUTPUT>] [<OTHER>]

--mode projects --add <SET> [<OTHER>]

--mode projects --delete <FILTER> [<OTHER>]

--mode projects --update <FILTER> <SET> [<OTHER>]

--mode projects --update <FILTER> <REMOVE> [<OTHER>]
```

Projects mode options

In general, you can specify options in any order. The exception is when you add more than one project within a single command. In this case, you must specify the options for the properties of each new project at the same time.

The <OTHER> option listed in the synopsis refers to sets of command options that are common to all modes. The options are:

- Common OUTPUT options
- CONNECTION options
- Shared options

OPERATION options
Specify exactly one OPERATION option.

--show
Output a comma separated value (CSV) list of projects that match the filter criteria. Use the `--fields` option to control the display of the project fields and their order. FILTER options are not required with `--show`.

--add
Add one or more new projects. A project name is required for each project that is added. You can add multiple projects with a single command line by specifying groups of SET options for each new project.

--delete
Delete the project or projects that match the filter criteria. At least one FILTER option is required.

--update
Update the attributes of projects that match the filter criteria. At least one FILTER option and one SET/REMOVE option is required.

FILTER options
FILTER options focus the set of projects that are operated on. You can specify multiple instances of each filter, except for `--name` and `--description`.

If you specify multiple instances of the same filter, the values are effectively ORed together. Different filter options are effectively ANDed together. For example, `--stream a --stream b --description c --description d` matches projects of the criteria `(stream = a OR b) AND (description = c OR d)`.

The FILTER options are:

--name <glob>
Operate on a project, or projects, whose name matches the specified glob pattern.

--description|--desc <glob>
Operate on a project, or projects, whose description matches the specified glob pattern.

--stream <glob>
Operate on projects that are associated with a stream whose name matches the glob pattern.

OUTPUT options
OUTPUT options are not required and are only valid with the `--show` operation.

The OUTPUT options are:

--output fields
: Display the list of valid field names for the Projects mode. The field names can then be used with the `--fields` option.

--output streams
: Display information about each stream associated with the project, rather than the information about the project itself.

Output Options
: You can also specify Output options that are common to all modes.

SET options
: The SET options apply changes to project attributes. Use `--add` to set the attributes of new projects, or `--update` to update the attributes of existing projects. At least one SET option is required with `--update`.

--set name:<name>
: Specify a name for a new project with the `--add` OPERATION option. This option is required for each project being added with `--add`. For example:

```
--add --set name:Project1
```

Update the name of an existing project using the `--update` OPERATION option. For example:

```
--update --name A --set name:B
```

Project names must be between 1 and 256 characters and are case insensitive. Project names can not contain the following characters:

- `:` (colon)
- * (asterisk)
- `/` (forward slash)
- \ (back slash)
- `` ` `` (backtick)
- `'` (single quote)
- " (double quote)

--set {description|desc}:<description>
: Specify a description for a new project using the `--add` OPERATION option, or update the description of an existing project using the `--update` OPERATION option.

--insert stream:<name>
: Associate an existing stream with a new project using the `--add` OPERATION option, or with an existing project using the `--update` OPERATION option.

This option does not create the specified stream, or streams. The streams must already exist in Coverity Connect. You can specify multiple `--insert stream` options to associate multiple streams with a project with a single command.

A stream has two types of associations with a project:

- Primary, in which a given stream is associated with a designated primary project.
- Linked, in which a non-primary project is associated with the given project through a stream link.

See Default Triage Scope for more information about primary projects and stream links.

When a stream is associated with a primary project (`ProjectA`) and then inserted into another project (`ProjectB`), its association with `ProjectA` is changed from a primary association to a stream link. `ProjectB` becomes the stream's primary project. Although `cov-manage-im` cannot explicity create stream links, this mechanism can be used to create a stream link.

To help you locate primary and linked associations, stream listings in Streams mode have a column called `Primary Project`. This column contains the name of the primary project that is associated with the stream (if any). Additionally, in Projects mode, you can specify `is-stream-linked` in the `--fields` option. This produces a column that displays `yes` if the stream has a linked association, or `no` if it has a primary association.

REMOVE options

The REMOVE options remove stream associations from projects. Remove options work for both primary and linked streams.

These options are only valid with the `--update` OPERATION option.

At least one FILTER option must be specified to prevent accidental bulk removals.

The REMOVE options are:

--remove stream:<name>

Remove a stream association from the specified projects. Streams are not removed from the Coverity Connect. Only the project's association with the stream is removed.

The <name> argument must exactly match a stream name.

You can specify multiple `--remove stream` options.

--clear streams

Remove all stream associations from the selected project(s).

Projects mode examples

The first example shows the four connection options that must contain values either on the command line, or in the XML configuration file (host, port, user, password). These connection options are intentionally dropped from most of the subsequent examples to reduce the length of the command lines.

When the connection options are not specified, assume that the values are retrieved from the default XML configuration file.

Show examples

Show all projects.

```
> cov-manage-im --host cim.company.com --port 8080 --user test \
    --password secret --mode projects --show
```

Show all projects that contain a stream named `x` or `y`.

```
> cov-manage-im --mode projects --show --stream x --stream y
```

List the fields that can be passed to `--fields` in projects mode.

```
> cov-manage-im --mode projects --show --output fields
```

Next, control what fields are shown by specifying some of these fields with the `--fields` option.

```
> cov-manage-im --mode projects --show --fields project
```

Show stream associations for all projects.

```
> cov-manage-im --mode projects --show --output streams
```

Show stream associations for projects where the project name starts with 'a'.

```
> cov-manage-im --mode projects --show --output streams --name "a*"
```

Add examples

Add a new project with minimal attributes specified.

```
> cov-manage-im --mode projects --add --set name:"HelloWorld"
```

Add a new project with all attributes specified

```
> cov-manage-im --mode projects --add \
    --set name:"hello world" \
    --set desc:"A full project" \
    --insert stream:mystream
```

Add two new projects at the same time.

```
> cov-manage-im --mode projects --add \
    --set name:proj1 \
    --set desc:"First project" \
    --insert stream:mystream   \
    --set name:proj2 \
    --set desc:"Second project"
```

Delete examples

Delete a project named `old-project.`

```
> cov-manage-im --mode projects --delete --name old-project
```

Update examples

Rename project A to B and update description at the same time.

```
> cov-manage-im --mode projects --update --name A \
    --set name:B --set "desc:This is now a B project"
```

Add stream associations to project P's existing stream associations.

```
> cov-manage-im --mode projects --update --name P \
    --insert stream:x
```

Remove all stream associations named x from project P.

```
> cov-manage-im --mode projects --update --name P --remove stream:x
```

Remove all stream associations from project P

```
> cov-manage-im --mode projects --update --name P \
    --clear streams
```

Streams mode

Query, add, delete, and update streams in Coverity Connect.

Synopsis

```
--mode streams --show [<FILTER>] [<OUTPUT>] [<OTHER>]

--mode streams --add <SET> [<OTHER>]

--mode streams --delete <FILTER> [<OTHER>]

--mode streams --update <FILTER> <SET> [<OTHER>]
```

Streams mode options

In general, you can specify options in any order. The exception is when you add more than one stream within a single command. In this case, you must specify the options for the properties of each new stream at the same time.

The <OTHER> option listed in the synopsis refers to sets of command options that are common to all modes. The options are:

- Common OUTPUT options
- CONNECTION options
- Shared options

OPERATION options

Specify exactly one OPERATION option in Streams mode.

--show

Output a comma separated value (CSV) list of streams that match the filter criteria. Use the `--fields` option to control the display of the stream fields and their order. FILTER options are not required with `--show`.

--add

Add new streams. A minimum of one stream name is required for each stream that you add.

You can add multiple streams with a single command by specifying groups of SET options for each new stream.

--delete

Delete the streams that match the filter criteria. At least one FILTER option is required.

--update

Update the name or description of the streams that match the filter criteria. A stream's language can not be updated after a stream has been created. At least one FILTER option and one SET option is required.

FILTER options

FILTER options focus the set of streams that are operated on. You can specify multiple instances of each filter, except for `--name` and `--description`.

If you specify multiple instances of the same filter, the values are effectively ORed together. Different filter options are effectively ANDed together. For example, `--project a --project b --language c --language d` matches streams of the criteria `(project = a OR b) AND (language = c OR d)`.

--name <glob>

Operate on a stream, or streams, whose name matches the glob pattern.

--language|--lang <lang>

Operate on a stream, or streams, based on language. `lang` can be one of:

- `C/C++, cpp`
- `Java, java`
- `C#, cs`
- `dynamic_java`
- `Mixed, mixed`
- Other, other

--description|--desc <glob>

Operate on a stream, or streams, whose description matches the glob pattern.

--stream <glob>
Operate on a stream, or streams, whose name matches the glob pattern.

--project <glob>
Operate on streams that are associated with projects whose name matches the glob pattern.

OUTPUT options
OUTPUT options are optional and are valid only with the `--show` option.

--output fields
Display the list of valid field names for this mode. These field names can then be used with the `--fields` option.

Output Options
You can also specify Output options that are common to all modes.

SET options
The SET options apply changes to stream attributes. Use `--add` to set the attributes of new streams, or `--update` to update the attributes of existing streams. At least one SET option is required with `--update`.

--set name:<name>
Specify a name for a new stream with the `--add` OPERATION option. This option is required for each stream added with `--add`. For example

```
--add --set name:Stream1
```

Update the name of an existing stream using the `--update` OPERATION option. For example:

```
--update --name A --set name:B
```

Stream names must be between 1 and 256 characters and are case sensitive. Stream names can not contain the following characters:

- `:` (colon)
- * (asterisk)
- `/` (forward slash)
- \ (backslash)
- ` (backtick)
- ' (single quote)
- " (double quote)

--set {language|lang}: <lang>
Specify a language for a new stream using the `--add` OPERATION option. `--set language` can be used for each stream you add. If you do not set a language for the stream, it defaults

to `mixed`. You can not update the language of an existing stream with this option. You should choose the default (mixed) for each new stream that you create. The other languages are provided for backward compatibility for previously released Coverity Connect versions (in which implicitly designated languages were required for streams). The valid languages are:

- `Mixed, mixed`
- `C/C++, cpp`
- `Java, java`
- `C#, cs`
- `dynamic_java`
- `Other, other`

Note

`Other` can be used when creating a stream with the `--add` option. `Other` is among the languages that may be shown when displaying a stream with the `--show` option.

--set {description|desc}:<description>
Specify a description for a new stream using the `--add` OPERATION option, or update the description of a stream using the `--update` OPERATION option.

--set {component-map|cmap}:<component-map>
Specify a component map to associate with a stream using the `--add` OPERATION option.

--set triage:<triage-store>
Specify a triage store to which the stream will belong. This option must be specified to create a new stream.

To use the default, rather than choosing a specific triage store when creating a stream, use `--set triage:"Default Triage Store"`.

--set expiration:{disabled|enabled}
This option allows you to set Coverity Connect to automatically delete streams after a period of inactivity. Only streams that are specifically configured for this feature are eligible for automatic deletion.

To set this option for newly created streams, use `--add`, for example `--mode streams --add --set expiration:enabled`.

To set this option for existing streams use `--update`, for example `--mode streams --update --set expiration:enabled`.

These options are disabled by default.

For more information, see "Designating a stream for auto-deletion of expired streams" in the *Coverity Platform 8.0 User and Administrator Guide*

--set desktopAnalysis:{disabled|enabled}
This option allows (or prohibits) the stream to provide data for Desktop Analysis. Only streams that have specifically enabled Desktop Analysis can be used as reference streams for Desktop Analysis users.

To set this option for newly created streams, use `--add`, for example `--mode streams --add --set desktopAnalysis:enabled`.

To set this option for existing streams use `--update`, for example `--mode streams --update --set desktopAnalysis:enabled`.

These options are disabled by default.

For more information, see the *Coverity Platform 8.0 User and Administrator Guide*

Streams mode examples

The first example shows the four connection options that must contain values either on the command line, or in the XML configuration file (host, port, user, password). These connection options are intentionally dropped from most of the subsequent examples to reduce the length of the command lines. When the connection options are not specified, assume that the values are retrieved from the default XML configuration file.

Note

The name of the built-in triage store is *Default Triage Store*. Use this triage store unless a different one has been set up.

Show examples

Show all streams.

```
> cov-manage-im --host cim.company.com --port 8080 --user test \
    --password secret --mode streams --show
```

Show all streams whose name starts with "linux-".

```
> cov-manage-im --mode streams --show --name "linux-*"
```

Show all streams whose language is Java.

```
> cov-manage-im --mode streams --show --lang Java
```

List the fields that can be passed to `--fields` in streams mode.

```
> cov-manage-im --mode streams --show --output fields
```

Control the fields that are shown by specifying some of these fields with the `--fields` option.

```
> cov-manage-im --mode streams --show  \
       --fields stream,language,desktop-analysis
```

Displays the attributes of stream `mystream`, including the stream's language, `other`.

```
> cov-manage-im --mode streams --show  \
        --name mystream
```

Add examples

Add a new C/C++ stream with minimal attributes specified.

```
> cov-manage-im --mode streams --add \
    --set name:HelloWorld \
    --set lang:mixed \
    --set triage:mytriagestore
```

Add a new stream with all attributes specified, and desktop analysis enabled.

```
> cov-manage-im --mode streams --add \
    --set name:HelloWorld \
    --set lang:mixed \
    --set triage:mytriagestore \
    --set "desc:My stream" \
    --set desktopAnalysis:enabled
```

Add two new streams at the same time

```
> cov-manage-im --mode streams --add \
    --set name:stream1 \
    --set lang:mixed \
    --set desc:"My new stream" \
    --set triage:mytriagestore \
    --set name:stream2 \
    --set desc:"My other new stream" \
    --set triage:mytriagestore
```

Associate the `stream1` stream with the `component1` component map.

```
> cov-manage-im --mode streams --update --name stream1 \
    -- set component-map:component1
```

Adds a new stream with `other`.

```
> cov-manage-im --mode streams --add --set --name:mystream \
    lang:other --set triage:mytriagestore
```

Delete examples

Delete the stream named `old-stream` (it can only be deleted if defects have NOT yet been committed).

```
> cov-manage-im --mode streams --delete --name old-stream
```

Update example

Rename stream `A` to `B` and update description at the same time.

```
> cov-manage-im --mode streams --update --name A \
    --set name:B --set "desc:This is now a B stream"
```

Triage mode

Query, add, delete, and update triage stores in Coverity Connect.

Synopsis

```
--mode triage --show [<FILTER>][<OUTPUT>] [<OTHER>]

--mode triage --add <SET> [<OTHER>]

--mode triage --delete <FILTER> [<OTHER>]

--mode triage --update <FILTER> <SET> [<OTHER>]
```

Triage mode options

In general, you can specify options in any order. The exception is when you add more than one triage store within a single command. In this case, you must specify the options for the properties of each new triage store at the same time.

The <OTHER> option listed in the synopsis refers to sets of command options that are common to all modes. The options are:

- Common OUTPUT options
- CONNECTION options
- Shared options

OPERATION options
Specify exactly one OPERATION option in Triage mode.

--show
Output a comma separated value (CSV) list of triage stores and their descriptions that match the filter criteria. Use the `--fields` option to control the display of the triage store fields and their order. FILTER options are not required with `--show`.

--add
Add new triage stores. A minimum of one triage store name is required for each triage store that you add.

You can add multiple triage stores with a single command by specifying groups of SET options for each new triage stores.

--delete
Delete the triage stores that match the name you specify.

--update
Update the name or description of the triage store that match the filter criteria. At least one FILTER option and one SET option is required.

FILTER options
FILTER options focus the set of triage stores that are operated on. You can specify multiple instances of each filter.

--name <glob>
Operate on a triage store, or triage stores, that match the name.

SET options
The SET options apply changes to triage store names and descriptions. Use `--add` to create a new triage store or description, or `--update` to update the triage store. At least one SET option is required with `--update`.

--set name:<name>
Specify a name for a new triage store with the `--add` OPERATION option. This option is required for each triage store added with `--add`. For example

```
--add --set name:triagestore1
```

Update the name of an existing triage store using the `--update` OPERATION option. For example:

```
--update --name triagestore1 --set name:triagestore2
```

Names must be between 1 and 256 characters and are case sensitive. Names can not contain the following characters:

- `:` (colon)
- * (asterisk)
- `/` (forward slash)
- \ (backslash)
- ` (backtick)
- ' (single quote)
- " (double quote)

--set {description|desc}:<description>
Specify an optional description for a new triage store using the `--add` OPERATION option, or update the description of a triage stream using the `--update` OPERATION option.

Triage mode examples

The first example shows the four connection options that must contain values either on the command line, or in the XML configuration file (host, port, user, password). These connection options are intentionally dropped from most of the subsequent examples to reduce the length of the command lines. When the connection options are not specified, assume that the values are retrieved from the default XML configuration file.

Note

The name of the built-in triage store is *Default Triage Store*. Use this triage store unless a different one has been set up.

Show examples

Show all triage stores and descriptions.

```
> cov-manage-im --host cim.company.com --port 8080 --user test \
    --password secret --mode triage --show
```

Show individual triage store named `mytriagestore` .

```
> cov-manage-im --mode triage --show --name mytriagestore
```

Add examples

Add a new triage store and description.

```
> cov-manage-im --mode triage --add \
    --set name:mytriagestore \
    --set desc:"This is my new triage store"
```

Update examples

Change a triage store name and its description.

```
> cov-manage-im --mode triage --update --name mytriagestore \
    --set name:yourtriagestore \
    --set desc:"This is your new triage store"
```

Delete example

Delete the store triage named `yourtriagestore` (it can only be deleted if it does contain any streams)

```
> cov-manage-im --mode triage --delete --name yourtriagestore
```

MOTD mode

Sets and gets a message of the day for a Coverity Connect instance.

Synopsis

```
--mode motd --show [<OUTPUT>] [<OTHER>]

--mode motd --update <SET> [<OTHER>]
```

MOTD mode options

In general, you can specify options in any order.

The `<OTHER>` option listed in the synopsis refers to sets of command options that are common to all modes. The options are:

- Common OUTPUT options
- CONNECTION options
- Shared options

OPERATION options
Specify exactly one OPERATION option in motd mode.

--show
Outputs the current message of the day.

--update
Updates the message of the day.

OUTPUT options
OUTPUT options are optional and are valid only with the `--show` option.

-no-headers
Displays the message of the day without a header if `enabled` is specified.

SET options
The SET options apply changes the message of the day. Use `--update` to update the message of the day. At least one SET option is required with `--update`.

--set message:"<message>"
Specify a message. For example

```
--set --message:"this is the message of the day"
```

MOTD mode examples

The first example shows the four connection options that must contain values either on the command line, or in the XML configuration file (host, port, user, password). These connection options are intentionally dropped from most of the subsequent examples to reduce the length of the command lines. When the connection options are not specified, assume that the values are retrieved from the default XML configuration file.

Show examples

Show message of the day.

```
> cov-manage-im --host cim.company.com --port 8080 --user test \
    --password secret --mode motd --show -no-headers
```

Update example

Change the message of the day.

```
> cov-manage-im --mode motd --update --set message:"hello, Hello, HELLO"
```

Commit mode

Gets, enables, and disables the commit gate, which permits or prevents commits to the Coverity Connect database.

Synopsis

```
--mode commit --update --set <enabled>|<disabled> [<OTHER>]
```

Commit mode options

In general, you can specify options in any order.

The <OTHER> option listed in the synopsis refers to sets of command options that are common to all modes. The options are:

- Common OUTPUT options
- CONNECTION options
- Shared options

OPERATION options
: Specify exactly one OPERATION option in commit mode.

  --update
  : Updates the commit gate status.

SET options
: The SET options apply changes to the commit gate. Use `--update` to update the commit gate. At least one SET option is required with `--update`.

  --set stats:{enabled | disabled}
  : Opens the commit gate (enabled) or closes the commit gate (disabled).

Commit mode examples

The first example shows the four connection options that must contain values either on the command line, or in the XML configuration file (host, port, user, password). These connection options are intentionally dropped from most of the subsequent examples to reduce the length of the command lines. When the connection options are not specified, assume that the values are retrieved from the default XML configuration file.

Update examples

Enable the commit gate.

```
> cov-manage-im --mode commit --update --set status:enabled
```

Disable the commit gate.

```
> cov-manage-im --mode commit --update --set status:disabled
```

Notification mode

Manually triggers a notification on a specific Coverity Connect view.

Synopsis

```
--mode notification --execute --view <viewName>
```

For shared views, only the user who shared the view can trigger the notification. An error will occur if the user with whom the view is shared attempts to trigger the notification through this command option.

For help configuring notification settings for a view in Coverity Connect, see *Coverity Platform 8.0 User and Administrator Guide*.

Notification mode options

In general, you can specify options in any order.

The options are:

- Common OUTPUT options
- CONNECTION options
- Shared options

OPERATION options
: Specify exactly one OPERATION option in commit mode.

  --execute
  : Triggers the firing of the notification email. `--execute` requires the inclusion of the `--view` filter.

FILTER options
: FILTER options focus the set of defects that are operated on.

  --view `<viewName>`
  : Specifies the Coverity Connect view for which the notification will be sent.

Notification mode example

This example shows the command to trigger a notification on the Coverity Connect view, *myView*.

```
> cov-manage-im --mode notification --execute --view myView
```

Authentication key mode

Create or revoke an authentication key for secure communication with the Coverity Connect server.

Synopsis

```
--mode auth-key --create --output-file <filename> [<SET>]

--mode auth-key --revoke <auth-key-ID>
```

Authentication key mode options

In general, you can specify options in any order.

SET options
The SET options apply certain attributes to the authentication key created.

--set description:<description>
Sets the description of the authentication key. If not provided, the description is an empty string.

--set expiration:<dateTime>
Sets the expiration date for the authentication key. There are four accepted syntaxes for <dateTime>:

- YYYY-MM-DD

  The authentication key will expire on the date specified.

- YYYY-MM-DD[T]hh:mm(:ss)

  The authentication key will expire on the date and time specified. Date and time may be separated by "T" or a space, and seconds are optional.

- "after_<N>_days"

  The authentication key will expire <N> days in the future.

- "after_<N>.<M>_days"

  The authentication key will expire <N>.<M> days in the future.

Note

See the *Coverity Platform 8.0 User and Administrator Guide* for important information about authentication key restrictions.

Authentication key mode examples

Create example

This example creates an authentication file named "myFile" with the description, "test user authentication file." This authentication file will expire in 90 days.

```
> cov-manage-im --host cim.company.com --port 8080 --user test \
    --password secret --mode auth-key --create --output-file myFile \
    --set description:"test user authentication file" \
    --set expiration:"after_90_days"
```

Revoke example

This example revokes the authentication key with ID 12345.

```
> cov-manage-im --mode auth-key --revoke 12345
```

Common OUTPUT options

The following OUTPUT options are common to all modes.

--fields
Specify the fields to output with a `--show` OPERATION option. The list of fields are specified as a comma-separated value (CSV) list.

To display a list of field names that are valid for a given mode, use `--show --output fields` in that mode. For example:

```
> cov-manage-im --mode projects --show --output fields
```

The following examples generate a comma-separated list of the values in the specified fields.

```
> cov-manage-im --mode projects \
    --show --fields project,description,creation-date,last-modified-date
```

```
> cov-manage-im --mode projects  \
    --show --output streams --fields project,stream-name,is-stream-linked
```

```
> cov-manage-im --mode streams \
    --show --fields stream,language,description
```

```
> cov-manage-im --mode defects --stream MySampleStream \
    --show --fields action,cid,checker
```

The order in which the fields are specified in the CSV list is the order in which they display. The same field can be listed multiple times.

--no-headers|-nh
Do not print field headers with `--show` operation.

--separator <sep>
Use <sep> to separate CSV values instead of a comma.

--output-file|-of <file>
Write output to a file named by <file>.

CONNECTION options

The following CONNECTION options are common to all modes. You can also store connection details in the <install_dir_sa>/config/coverity_config.xml [p. 283] file.

--auth-key-file <keyfile>
Specify the location of your authentication key file, created in Authentication key mode. See the *Coverity Platform 8.0 User and Administrator Guide* for important information about authentication key restrictions.

--certs <filename>
In addition to CA certificates obtained from other trust stores, use the CA certificates in the given <filename>. For information on the new SSL certificate management functionality, please see *Coverity Platform 8.0 User and Administrator Guide*

--host <server-hostname>
Specify the Coverity Connect server hostname. To use this option, the Coverity Connect server must be running. If this option is unspecified, the default is the value from the *cim/host* element from the XML configuration file.

--on-new-cert <trust | distrust>
Indicates whether to trust (with trust-first-time) self-signed certificates, presented by the server, that the application has not seen before. For information on the new SSL certificate management functionality, please see *Coverity Platform 8.0 User and Administrator Guide*

--port <server-port>
Specify the Coverity Connect server HTTP or HTTPS connection port. To use this option, the Coverity Connect server must be running. If this option is unspecified, the default is established in the following order:

1. The value from the *cim/port* element from the XML configuration file.
2. *8080*. If `--ssl` is present, the default is *8443*.

--ssl
Allow Coverity Connect to use an SSL-encrypted channel.

--user <user_name>
Connect to Coverity Connect as `user_name`. If this option is unspecified, the default is established in the following order:

1. The *cim/client_security/user* element from the XML configuration file.
2. The *COV_USER* environment variable.
3. The *USER* environment variable.
4. The name of the operating system user invoking the command (if supported).

--password <password>
Specify the password for either the current user name, or the user specified with the `--user` option. If this option is unspecified, the default is established from the *cim/client_security/password* element from the XML configuration file.

--userLdapServer <domain>
Specify the domain of the user. If this option is not specified, the domain is resolved following this procedure:

1. If the user name contains "@", two possible users are considered, one with the name as given, and one with the name comprising the substring after the last "@".
2. If one and only one user name is found, then the domain is set to the domain of this user.
3. Otherwise, an error is output explaining that the domain could not be automatically set, and asking the user to explicitly specify a domain with —userLdapServer for users who are ldap users.

Note

Coverity recommends using this parameter to avoid issues when LDAP authentication is used.

Shared options

The following options are common to all modes of the `cov-manage-im` command.

--config|-c `<coverity_config.xml>`
Use a specified XML configuration file instead of the default configuration file located at `<install_dir_sa>/config/coverity_config.xml`.

--debug|-g
Turn on basic debugging output. You must specify a mode and at least one option for proper debug reporting. This option will issue a warning if the command line that you are using is not valid; for example, if you use an unsupported option in a particular mode.

--verbose|-V `<0, 1, 2, 3, 4>`
Set the detail level of command messages. Higher is more verbose (more messages). Defaults to 1. Use `--verbose 0` to disable the output of `--show` options.

--response-file|-rf `<file>`
Specify command line options in `<file>`. These options are processed as if they are specified on the command line at the same point as the response file is specified. Multiple response files can be used on a single command line, but nested response files are not allowed.

Lines in response files starting with # are considered to be comments and are ignored.

Name

cov-start-im Start Coverity Connect.

Synopsis

```
cov-start-im
```

Description

The `cov-start-im` command starts Coverity Connect.

On Windows systems, when Coverity Connect is installed as a service, the `cov-start-im.exe` program is often unnecessary because Coverity Connect starts automatically when the system boots up. When Coverity Connect is installed as a service, any administrator can use this command.

When Coverity Connect is not installed as a service, only the user who installed Coverity Connect is able to use `cov-start-im` to start it.

Name

cov-stop-im Stop Coverity Connect.

Synopsis

```
cov-stop-im
```

Description

The `cov-stop-im` command stops Coverity Connect.

On Windows systems, when Coverity Connect is installed as a service, the `cov-stop-im.exe` program is often unnecessary because Coverity Connect stops automatically when the system shuts down. When Coverity Connect is installed as a service, any administrator can use this command.

When Coverity Connect is not installed as a service, only the user who installed Coverity Connect is able to use `cov-stop-im` to stop it.

Name

cov-support Create support information package for Coverity Connect.

Synopsis

```
cov-support [-v | -vv] [--coverity-home <coverity-base-directory>] [--with-
config] [--with-logs [days]] -ol--output <outputfile>
```

Description

The `cov-support` command creates a compressed archive containing various property and log files for Coverity Connect. Support may request that you create this archive and submit it for analysis.

If `--with-config` is specified, then the following files in the `/config` folder are included in the archive:

- `cim.properties`
- `server.xml`
- `VERSION`
- `system.properties`
- `postgresql.conf`
- `web.properties`

If `--with-logs` is specified, then log files in the `/logs`, `/postgressql`, and `/.install4j` directories are added to the archive.

Options

--coverity-home `name_of_directory`
: Specify the directory where Coverity Connect is installed, if it is installed in a directory other than the directory specified by coverity-base-directory.

-o | --output `name_of_file`
: Destination output file. File is in tar-bzip2 format.

--with-config
: Include configuration files in the support archive.

--with-logs `<days>`
: Include log files in the support archive for the past number of days.

-v
: Enable verbose logging information (for debug purposes)

-vv
: Enable very verbose logging information (for trace purposes)

Example

To obtain the config files and the log files for five days, and put them in an archive named `support-archive`:

```
> cov-support -v --with-config  --with-logs 5 -o support-archive
```

Coverity Security Report

Name

cov-generate-security-report Generate a security report from an existing security report configuration.

Synopsis

```
cov-generate-security-report <configuration file> [--help] --output <output-
file> --password <spec>
```

Description

The `cov-generate-security-report` command creates an updated Security Report based on a configuration file and current issue data in Coverity Connect.

Options

<configuration file>
: Coverity Security Report configuration (.covsr) file

--help
: Display the help message and exit.

--output <output-file>
: Name of the PDF output file. The file will be replaced if present.

--password <spec>
: Coverity Connect password specifier

The password `<spec>` argument is required, and has four forms:

console
: The password is read from the keyboard without echoing to the console.

file:<filename>
: The password is read from the first line of the file <filename>.

file: -
: The password is read from the standard input. This is for use with pipes and redirection, not for keyboard input.

env:<variable>
: The password is read from the environment variable <variable>.

Examples

```
cov-generate-security-report MyConfiguration.covsr --output MyReport.pdf --password
 console
```

This command will use the configuration file MyConfiguration.covsr to generate a Security Report in the file MyReport.pdf. You will be prompted to enter the Coverity Connect server password on the console.

Coverity MISRA Report

Name

cov-generate-misra-report Generate a MISRA report from an existing MISRA report configuration.

Synopsis

```
cov-generate-misra-report <configuration file> [--help] --output <output-
file> --password <spec> --certs <certificate file> --on-new-cert <trust |
distrust>
```

Description

The `cov-generate-misra-report` command creates an updated MISRA Report based on a configuration file and current issue data in Coverity Connect.

Options

--certs <`filename`>
: In addition to CA certificates obtained from other trust stores, use the CA certificates in the given `filename`.

<configuration file>
: Coverity MISRA Report configuration (.covmr) file

--help
: Display the help message and exit.

--on-new-cert >trust | distrust>
: Indicates with `--ssl` whether to trust (with trust-first-time) self-signed certificates, presented by the server, that the application has not seen before.

--output <output-file>
: Name of the PDF output file. The file will be replaced if present.

--password <spec>
: Coverity Connect password specifier

The password <`spec`> argument is required, and has four forms:

console
: The password is read from the keyboard without echoing to the console.

file:<filename>
: The password is read from the first line of the file <filename>.

file:-
: The password is read from the standard input. This is for use with pipes and redirection, not for keyboard input.

env:<variable>
: The password is read from the environment variable <variable>.

Examples

```
cov-generate-misra-report MyConfiguration.covmr --output MyReport.pdf --password
 console
```

This command will use the configuration file MyConfiguration.covmr to generate a MISRA Report in the file MyReport.pdf. You will be prompted to enter the Coverity Connect server password on the console.

Accepted date/time formats

Appendix A. Coverity Glossary

Glossary

A

Abstract Syntax Tree (AST)	A tree-shaped data structure that represents the structure of concrete input syntax (from source code).
action	In Coverity Connect, a customizable attribute used to triage a CID. Default values are Undecided, Fix Required, Fix Submitted, Modeling Required, and Ignore. Alternative custom values are possible.
advanced triage	In Coverity Connect, streams that are associated with the same always share the same triage data and history. For example, if Stream A and Stream B are associated with Triage Store 1, and both streams contain CID 123, the streams will share the triage values (such as a shared *Bug* classification or a *Fix Required* action) for that CID, regardless of whether the streams belong to the same project. Advanced triage allows you to select one or more triage stores to update when triaging a CID in a Coverity Connect project. Triage store selection is possible only if the following conditions are true: • Some streams in the project are associated with one triage store (for example, TS1), and other streams in the project are associated with another triage store (for example, TS2). In this case, some streams that are associated with TS1 must contain the CID that you are triaging, and some streams that are associated with TS2 must contain that CID. • You have permission to triage issues in more than one of these triage stores. In some cases, advanced triage can result in CIDs with issue attributes that are in the Various state in Coverity Connect. See also, triage.
annotation	For C/C++, a comment with specific syntax in the source code that suppresses a false positive or enhances a function. For Java, the standard Java annotation format is supported for this purpose. Attributes to suppress false positives are not implemented for C# checkers. See also code annotation and function annotation.

C

call graph	A graph in which functions are nodes, and the edges are the calls between the functions.
category	See issue category.
checker	A program that traverses paths in your source code to find specific issues in it. Examples of checkers include RACE_CONDITION, RESOURCE_LEAK, and INFINITE_LOOP.
checker category	See issue category.
churn	A measure of change in defect reporting between two Coverity Analysis releases that are separated by one minor release, for example, 6.5.0 and 6.6.0.
CID (Coverity identifier)	See Coverity identification (CID).
classification	A category that is assigned to a software issue in the database. Built-in classification values are Unclassified, Pending, False Positive, Intentional, and Bug. For Test Advisor issues, classifications include Untested, No Test Needed, and Tested Elsewhere. Issues that are classified as Unclassified, Pending, and Bug are regarded as software issues for the purpose of defect density calculations.
code annotation	For C/C++, an annotation that suppresses a false positive. The analysis engine ignores events that are preceded by a code annotation, and defects that are caused by ignored events have the Intentional status in the Coverity Connect unless overridden. See also annotation and function annotation. For Java, the standard Java annotation format is supported for this purpose. Attributes to suppress false positives are not implemented for C# checkers.
code base	A set of related source files.
code coverage	The amount of code that is tested as a percentage of the total amount of code. Code coverage is measured different ways: line coverage, path coverage, statement coverage, decision coverage, condition coverage, and others.
component	A named grouping of source code files. Components allow developers to view only issues in the source files for which they are responsible, for example. In Coverity Connect, these files are specified by a Posix regular expression. See also, component map.

component map	Describes how to map source code files, and the issues contained in the source files, into components.
control flow graph	A graph in which blocks of code without any jumps or jump targets are nodes, and the directed edges are the jumps in the control flow between the blocks. The entry block is where control enters the graph, and the exit block is where the control flow leaves.
Coverity identifier (CID)	An identification number assigned to a software issue. A snapshot contains issue *instances* (or occurrences), which take place on a specific code path in a specific version of a file. Issue instances, both within a snapshot and across snapshots (even in different streams), are grouped together according to similarity, with the intent that two issues are "similar" if the same source code change would fix them both. These groups of similar issues are given a numeric identifier, the CID. Coverity Connect associates triage data, such as classification, action, and severity, with the CID (rather than with an individual issue).
CWE (Common Weakness Enumeration)	A community-developed list of software weaknesses, each of which is assigned a number (for example, see CWE-476 at http://cwe.mitre.org/data/definitions/476.html). Coverity associates many categories of defects (such as "Null pointer dereferences") with a CWE number.
Cyclomatic Complexity (CCM)	The number of linearly independent execution paths through the functions in your source code. The higher the number, the more complex the function. For example, the complexity of a function with no control flow statements is 1 because there is only one possible execution path. However, an `if` statement introduces at least two possible paths, one if the statement is true, another if it is false.

D

data directory	The directory that contains the Coverity Connect database. After analysis, the `cov-commit-defects` command stores defects in this directory. You can use Coverity Connect to view the defects in this directory. See also intermediate directory.
deadcode	Code that cannot possibly be executed regardless of what input values are provided to the program.
defect	See issue.
deterministic	A characteristic of a function or algorithm that, when given the same input, will always give the same output.
dismissed issue	Issue marked by developers as *Intentional* or *False Positive* in the Triage pane. When such issues are no longer present in the latest snapshot of the code base, they are identified as *absent dismissed*.

domain	A combination of the language that is being analyzed and the type of analysis, either static or dynamic.
dynamic analysis	Analysis of software code by executing the compiled program. See also static analysis.
dynamic analysis agent	A JVM agent for Coverity Dynamic Analysis that instruments your program to gather runtime evidence of defects.
dynamic analysis stream	A sequential collection of snapshots, which each contain all of the issues that Coverity Dynamic Analysis reports during a single invocation of the Coverity Dynamic Analysis broker.

E

event	In Coverity Connect, a software issue is composed of one or more events found by the analysis. Events are useful in illuminating the context of the issue. See also issue.

F

false negative	A defect in the source code that is not found by Coverity Analysis.
false path pruning (FPP)	A technique to ensure that defects are only detected on feasible paths. For example, if a particular path through a method ensures that a given condition is known to be true, then the `else` branch of an `if` statement which tests that condition cannot be reached on that path. Any defects found in the `else` branch would be impossible because they are “on a false path”. Such defects are suppressed by a false path pruner.
false positive	A potential defect that is identified by Coverity Analysis, but that you decide is not a defect. In Coverity Connect, you can dismiss such issues as false positives. You might also use annotations (also called code annotations) in your source code to identify such issues as intentional during the source code analysis phase, prior to sending analysis results to Coverity Connect.
fixed issue	Issue from the previous snapshot that is not in the latest snapshot.
fixpoint	The Extend SDK engine notices that the second and subsequent paths through the loop are not significantly different from the first iteration, and stops analyzing the loop. This condition is called a fixpoint of the loop.
flow-insensitive analysis	A checker that is stateless. The abstract syntax trees are not visited in any particular order.
function annotation	An annotation that enhances or suppresses a function model. See also annotation and code annotation.

function model	A model of a function that is not in the code base that enhances the intermediate representation of the code base that Coverity Analysis uses to more accurately analyze defects.

I

impact	Term that is intended to indicate the likely urgency of fixing the issue, primarily considering its consequences for software quality and security, but also taking into account the accuracy of the checker. Impact is necessarily probabilistic and subjective, so one should not rely exclusively on it for prioritization.
inspected issue	Issue that has been triaged or fixed by developers.
intermediate directory	A directory that is specified with the `--dir` option to many commands. The main function of this directory is to write build and analysis results before they are committed to the Coverity Connect database as a snapshot. Other more specialized commands that support the `--dir` option also write data to or read data from this directory. The intermediate representation of the build is stored in `<intermediate_directory>/emit` directory, while the analysis results are stored in `<intermediate_directory>/output`. This directory can contain builds and analysis results for multiple languages (C/C++, C#, and Java code bases). See also data directory.
intermediate representation	The output of the Coverity compiler, which Coverity Analysis uses to run its analysis and check for defects. The intermediate representation of the code is in the intermediate directory.
interprocedural analysis	An analysis for defects based on the interaction between functions. Coverity Analysis uses call graphs to perform this type of analysis. See also intraprocedural analysis.
intraprocedural analysis	An analysis for defects within a single procedure or function, as opposed to interprocedural analysis.
issue	Coverity Connect displays three types of software issues: quality defects, potential security vulnerabilities, and test policy violations. Some checkers find both quality defects and potential security vulnerabilities, while others focus primarily on one type of issue or another. The Quality Advisor, Coverity Security Advisor, and Test Advisor dashboards in Coverity Connect provide high-level metrics on each type of issue. Note that this glossary includes additional entries for the various types of issues, for example, an inspected issue, issue category, and so on.

issue category

A string used to describe the nature of a software issue; sometimes called a "checker category" or simply a "category." The issue pertains to a subcategory of software issue that a checker can report within the context of a given domain.

Examples:

- `Memory - corruptions`
- `Incorrect expression`
- `Integer overflow Insecure data handling`

Impact tables in the *Coverity 8.0 Checker Reference* list issues found by checkers according to their category and other associated checker properties.

K

killpath

For Coverity Analysis for C/C++, a path in a function that aborts program execution. See `<install_dir_sa>/library/generic/common/killpath.c` for the functions that are modeled in the system.

For Coverity Analysis for Java, and similarly for C#, a modeling primitive used to indicate that execution terminates at this point, which prevents the analysis from continuing down this execution path. It can be used to model a native method that kills the process, like `System.exit`, or to specifically identify an execution path as invalid.

kind

A string that indicates whether software issues found by a given checker pertain to SECURITY (for security issues), QUALITY (for quality issues), TEST (for issues with developer tests, which are found by Test Advisor), or QUALITY/SECURITY. Some checkers can report quality and security issues. The Coverity Connect UI can use this property to filter and display CIDs.

L

latest state

A CID's state in the latest snapshot merged with its state from previous snapshots starting with the snapshot in which its state was 'New'.

local analysis

Interprocedural analysis on a subset of the code base with Coverity Desktop plugins, in contrast to one with Coverity Analysis, which usually takes place on a remote server.

local effect

A string serving as a generic event message that explains why the checker reported a defect. The message is based on a subcategory of software issues that the checker can detect. Such strings appear in the Coverity Connect triage pane for a given CID.

Examples:

- `May result in a security violation.`
- `There may be a null pointer exception, or else the comparison against null is unnecessary.`

long description

A string that provides an extended description of a software issue (compare with type). The long description appears in the Coverity Connect triage pane for a given CID. In Coverity Connect, this description is followed by a link to a corresponding CWE, if available.

Examples:

- `The called function is unsafe for security related code.`
- `All paths that lead to this null pointer comparison already dereference the pointer earlier (CWE-476).`

M

model

For Coverity Analysis for C/C++, a representation of each method in the application that is used for interprocedural analysis, created as each function is analyzed. For example, the model shows which arguments are dereferenced, and whether the function returns a null value.

For Coverity Analysis for Java, and similarly for C#, a representation of each method in the application that is used for interprocedural analysis. For example, the model shows which arguments are dereferenced, and whether the function returns a null value. Models are created as each function is analyzed.

N

native build

The normal build process in a software development environment that does not involve Coverity products.

O

outstanding issue

Issues that are uninspected and unresolved.

owner

User name of the user to whom an issue has been assigned in Coverity Connect. Coverity Connect identifies the owner of issues not yet assigned to a user as *Unassigned*.

P

postorder traversal	The recursive visiting of children of a given node in order, and then the visit to the node itself. Left sides of assignments are evaluated after the assignment because the left side becomes the value of the entire assignment expression.
project	In Coverity Connect, a specified set of related streams that provide a comprehensive view of issues in a code base.

R

resolved issues	Issues that have been fixed or marked by developers as *Intentional* or *False Positive* through the Coverity Connect Triage pane.
run	In Prevent releases 4.5.x or lower, a grouping of defects committed to the Coverity Connect. Each time defects are inserted into the Coverity Connect using the `cov-commit-defects` command, a new run is created, and the run ID is reported. See also snapshot

S

sanitize	To clean or validate tainted data to ensure that the data is valid. Sanitizing tainted data is an important aspect of secure coding practices to eliminate system crashes, corruption, escalation of privileges, or denial of service. See also tainted data.
severity	In Coverity Connect, a customizable property that can be assigned to CIDs. Default values are Unspecified, Major, Moderate, and Minor. Severities are generally used to specify how critical a defect is.
sink	Coverity Analysis for C/C++: Any operation or function that must be protected from tainted data. Examples are array subscripting, `system(), malloc()`. Coverity Analysis for Java: Any operation or function that must be protected from tainted data. Examples are array subscripting and the JDBC API `Connection.execute`.
snapshot	A copy of the state of a code base at a certain point during development. Snapshots help to isolate defects that developers introduce during development. Snapshots contain the results of an analysis. A snapshot includes both the issue information and the source code in which the issues were found. Coverity Connect allows you to delete a snapshot in case you committed faulty data, or if you committed data for testing purposes.

snapshot scope	Determines the snapshots from which the CID are listed using the *Show* and the optional *Compared To* fields. The show and compare scope is only configurable in the *Settings* menu in *Issues:By Snapshot* views and the snapshot information pane in the *Snapshots* view.
source	An entry point of untrusted data. Examples include environment variables, command line arguments, incoming network data, and source code.
static analysis	Analysis of software code without executing the compiled program. See also dynamic analysis.
store	A map from abstract syntax trees to integer values and a sequence of events. This map can be used to implement an abstract interpreter, used in flow-sensitive analysis.
stream	A sequential collection of snapshots. Streams can thereby provide information about software issues over time and at a particular points in development process.

T

tainted data	Any data that comes to a program as input from a user. The program does not have control over the values of the input, and so before using this data, the program must sanitize the data to eliminate system crashes, corruption, escalation of privileges, or denial of service. See also sanitize.
triage	The process of setting the states of an issue in a particular stream, or of issues that occur in multiple streams. These user-defined states reflect items such as how severe the issue is, if it is an expected result (false positive), the action that should be taken for the issue, to whom the issue is assigned, and so forth. These details provide tracking information for your product. Coverity Connect provides a mechanism for you to update this information for individual and multiple issues that exist across one or more streams. See also advanced triage.
type	A string that typically provides a short description of the root cause or potential effect of a software issue. The description pertains to a subcategory of software issues that the checker can find within the scope of a given domain. Such strings appear at the top of the Coverity Connect triage pane, next to the CID that is associated with the issue. Compare with long description. Examples: `The called function is unsafe for security related code`

	`Dereference before null check`
	`Out-of-bounds access`
	`Evaluation order violation`
	Impact tables in the *Coverity 8.0 Checker Reference* list issues found by checkers according to their type and other associated checker properties.

U

unified issue	An issue that is identical and present in multiple streams. Each instance of an identical, unified issue shares the same CID.
uninspected issues	Issues that are as yet unclassified in Coverity Connect because they have not been triaged by developers.
unresolved issues	Defects are marked by developers as *Pending* or *Bug* through the Coverity Connect Triage pane. Coverity Connect sometimes refers to these issues as *Outstanding* issues.

V

various	Coverity Connect uses the term *Various* in two cases: • When a checker is categorized as both a quality and a security checker. For example, USE_AFTER_FREE and UNINIT are listed as such in the *Issue Kind* column of the View pane. All C/C++ security checkers are also treated as quality checkers by Coverity Connect. For details, see the *Coverity 8.0 Checker Reference*. • When different instances of the same CID are triaged differently. Within the scope of a project, instances of a given CID that occur in separate streams can have different values for a given triage attribute if the streams are associated with different . For example, you might use advanced triage to classify a CID as a *Bug* in one triage store but retain the default *Unclassified* setting for the CID in another store. In such a case, the View pane of Coverity Connect identifies the project-wide classification of the CID as *Various*. Note that if all streams share a single triage store, you will never encounter a CID in this triage state.
view	Saved searches for Coverity Connect data in a given project. Typically, these searches are filtered. Coverity Connect displays this output in data tables (located in the Coverity Connect View pane). The columns in these tables can include CIDs, files, snapshots, checker names, dates, and many other types of data.

Appendix B. Legal Notice

The information contained in this document, and the Licensed Product provided by Synopsys, are the proprietary and confidential information of Synopsys, Inc. and its affiliates and licensors, and are supplied subject to, and may be used only by Synopsys customers in accordance with the terms and conditions of a license agreement previously accepted by Synopsys and that customer. Synopsys' current standard end user license terms and conditions are contained in the `cov_EULM` files located at `<install_dir>/doc/en/licenses/end_user_license`.

Portions of the product described in this documentation use third-party material. Notices, terms and conditions, and copyrights regarding third party material may be found in the `<install_dir>/doc/en/licenses` directory.

Customer acknowledges that the use of Synopsys Licensed Products may be enabled by authorization keys supplied by Synopsys for a limited licensed period. At the end of this period, the authorization key will expire. You agree not to take any action to work around or override these license restrictions or use the Licensed Products beyond the licensed period. Any attempt to do so will be considered an infringement of intellectual property rights that may be subject to legal action.

If Synopsys has authorized you, either in this documentation or pursuant to a separate mutually accepted license agreement, to distribute Java source that contains Synopsys annotations, then your distribution should include Synopsys' `analysis_install_dir/library/annotations.jar` to ensure a clean compilation. This `annotations.jar` file contains proprietary intellectual property owned by Synopsys. Synopsys customers with a valid license to Synopsys' Licensed Products are permitted to distribute this JAR file with source that has been analyzed by Synopsys' Licensed Products consistent with the terms of such valid license issued by Synopsys. Any authorized distribution must include the following copyright notice: **Copyright © 2016 Synopsys, Inc. All rights reserved worldwide**.

U.S. GOVERNMENT RESTRICTED RIGHTS: The Software and associated documentation are provided with Restricted Rights. Use, duplication, or disclosure by the U.S. Government is subject to restrictions set forth in subparagraph (c)(1) of The Rights in Technical Data and Computer Software clause at DFARS 252.227-7013 or subparagraphs (c)(1) and (2) of Commercial Computer Software – Restricted Rights at 48 CFR 52.227-19, as applicable.

The Manufacturer is: Synopsys, Inc. 690 E. Middlefield Road, Mountain View, California 94043.

The Licensed Product known as Coverity is protected by multiple patents and patents pending, including U.S. Patent No. 7,340,726.

Trademark Statement

> Coverity and the Coverity logo are trademarks or registered trademarks of Synopsys, Inc. in the U.S. and other countries. Synopsys' trademarks may be used publicly only with permission from Synopsys. Fair use of Synopsys' trademarks in advertising and promotion of Synopsys' Licensed Products requires proper acknowledgement.

Microsoft, Visual Studio, and Visual C# are trademarks or registered trademarks of Microsoft Corporation in the United States and/or other countries.

Microsoft Research Detours Package, Version 3.0.

This Licensed Product contains open source or community source software ("**Open Source Software**") provided under separate license terms (the "**Open Source License Terms**"), as described in the applicable license agreement under which this Licensed Product is licensed ("**Agreement**"). The applicable Open Source License Terms are identified in a directory named `licenses` provided with the delivery of this Licensed Product. For all Open Source Software subject to the terms of an LGPL license, Customer may contact Synopsys at `support@coverity.com` and Synopsys will comply with the terms of the LGPL by delivering to Customer the applicable requested Open Source Software package, and any modifications to such Open Source Software package, in source format, under the applicable LGPL license. Any Open Source Software subject to the terms and conditions of the GPLv3 license as its Open Source License Terms that is provided with this Licensed Product is provided as a mere aggregation of GPL code with Synopsys' proprietary code, pursuant to Section 5 of GPLv3. Such Open Source Software is a self-contained program separate and apart from the Synopsys code that does not interact with the Synopsys proprietary code. Accordingly, the GPL code and the Synopsys proprietary code that make up this Licensed Product co-exist on the same media, but do not operate together. Customer may contact Synopsys at `support@coverity.com` and Synopsys will comply with the terms of the GPL by delivering to Customer the applicable requested Open Source Software package in source code format, in accordance with the terms and conditions of the GPLv3 license. No Synopsys proprietary code that Synopsys chooses to provide to Customer will be provided in source code form; it will be provided in executable form only. Any Customer changes to the Licensed Product (including the Open Source Software) will void all Synopsys obligations under the Agreement, including but not limited to warranty, maintenance services and infringement indemnity obligations.

The Cobertura package, licensed under the GPLv2, has been modified as of release 7.0.3. The package is a self-contained program, separate and apart from Synopsys code that does not interact with the Synopsys proprietary code. The Cobertura package and the Synopsys proprietary code co-exist on the same media, but do not operate together. Customer may contact Synopsys at `support@coverity.com` and Synopsys will comply with the terms of the GPL by delivering to Customer the applicable requested open source package in source format, under the GPLv2 license. Any Synopsys proprietary code that Synopsys chooses to provide to Customer upon its request will be provided in object form only. Any changes to the Licensed Product will void all Coverity obligations under the Agreement, including but not limited to warranty, maintenance services and infringement indemnity obligations. If Customer does not have the modified Cobertura package, Synopsys recommends to use of the JaCoCo package instead.

For information about using JaCoCo, see the description for `cov-build --java-coverage` in the *Command and Ant Task Reference*.

LLVM/Clang subproject

Copyright © All rights reserved. Developed by: University of Illinois at Urbana-Champaign, Computer Science Department (`http://cs.illinois.edu/`). Permission is hereby granted, free of charge,

to any person obtaining a copy of LLVM/Clang and associated documentation files ("Clang"), to deal with Clang without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of Clang, and to permit persons to whom Clang is furnished to do so, subject to the following conditions: Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimers. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimers in the documentation and/or other materials provided with the distribution. Neither the name of the University of Illinois at Urbana-Champaign, nor the names of its contributors may be used to endorse or promote products derived from Clang without specific prior written permission.

CLANG IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE CONTRIBUTORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH CLANG OR THE USE OR OTHER DEALINGS WITH CLANG.

Rackspace Threading Library (2.0)

Copyright © Rackspace, US Inc. All rights reserved. Licensed under the Apache License, Version 2.0 (the "License"); you may not use these files except in compliance with the License. You may obtain a copy of the License at `http://www.apache.org/licenses/LICENSE-2.0`.

Unless required by applicable law or agreed to in writing, software distributed under the License is distributed on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied. See the License for the specific language governing permissions and limitations under the License.

SIL Open Font Library subproject

Copyright © 2016, Synopsys Inc. All rights reserved worldwide. (`www.coverity.com`), with Reserved Font Name fa-gear, fa-info-circle, fa-question.

This Font Software is licensed under the SIL Open Font License, Version 1.1. This license is available with a FAQ at `http://scripts.sil.org/OFL`.