Coverity Desktop 2020.12 for IntelliJ IDEA and Android Studio: User Guide

Table of Contents

Chapter 1. Coverity Desktop overview

Table of Contents

Coverity Desktop is a supplemental Coverity Analysis tool that allows you to view and manage issues in your source code within IntelliJ IDEA or Android Studio. Coverity Analysis results are displayed in the Coverity Desktop plug-in, allowing you to assess the risk, and fix issues in the IDE. Analysis results from your code base are displayed in the Coverity Desktop plug-in to help you find and report hard-to-spot software defects, potential security vulnerabilities, and test policy violations. Note that Dynamic Analysis defects are not displayed in Coverity Desktop.

Coverity Desktop provides two analysis modes:

- Central analysis
- Local analysis

After Coverity Desktop displays your issue data, you can view the issues for your project marked directly in the IDE's code editor view. Coverity Desktop provides two views: Issues and Details. Use these views to navigate through your issues, to determine the impact of the issues on your code, and to manage and update (*triage*) the state of each issue.

Installation instructions and supported platform details are located in the *Coverity 2020.12 Installation and Deployment Guide*.

Note

To view issue information, such as event messages and so forth, in Japanese, you must make sure that the locale is set for Japan in your Coverity Connect user preferences. For more information, see the *Coverity Platform 2020.12 User and Administrator Guide*.

1.1. Requirements

Coverity Desktop for IntelliJ IDEA/Android Studio has the following requirements (refer to the "Coverity Desktop" chapter of the *Coverity 2020.12 Installation and Deployment Guide* for supported IDE and Java version numbers):

- Coverity Connect 2020.12
- IntelliJ IDEA or Android Studio
- Java Runtime Environment

It is assumed that you have a functional knowledge of your IDE, as well as Coverity Analysis, Coverity Connect, and Test Advisor concepts. If you need more information, consult the following Coverity product documentation:

For more information about Coverity Connect, see the *Coverity Platform 2020.12 User and Administrator Guide*.

For more information about Coverity Analysis, see the *Coverity Analysis 2020.12 User and Administrator Guide*.

For more information about Test Advisor, see the *Test Advisor 2020.12 User and Administrator Guide*.

1.2. Coverity Desktop concepts and use cases

This section provides conceptual definitions and use cases for Coverity Desktop. The point of this chapter is to help you decide how you would like to configure and run the Coverity Desktop tools to accomplish your tasks. After you decide on an analysis mode, see the subsequent chapters in this guide for information about server and analysis configuration, running a local analysis, and examining and triaging issues within the IDE.

1.2.1. Concepts

Central Analysis. A central analysis occurs when the central source code repository is analyzed by Coverity Analysis outside of the IDE. Central analyses are typically run on a build server and are triggered by an automated process. In most installations, a central analysis will be run as part of a nightly build. Alternatively, a central analysis might be initiated whenever code is checked in to the source code repository.

The issues discovered by the central analysis are committed to a Coverity Connect database that Coverity Desktop uses to retrieve and update issue information that you view and triage within the IDE. Central analysis options, including server and stream information, are set using one or more Analysis Configurations. You must set up at least one analysis configuration to retrieve central analysis results in the plug-in.

Local Analysis. With local analysis, you analyze your source code for issues within your IDE. By running a local analysis, you can examine issues, fix issues, and verify that the issues are fixed before checking your code back into the central code repository. You have the option of analyzing a single source file, a set of source files of your choosing, or your entire project.

Local analysis options are set using one or more Analysis Configurations. Upon initial set up Coverity Desktop, you will have a default analysis configuration with your choice of analysis options. You may create additional analysis configurations, with different sets of options, and choose the correct configuration for each individual analysis run. This allows you to easily change the type of analysis you want to run, depending on the project or file(s) you are currently working on.

Before you begin using Coverity Desktop, there are a few important points to keep in mind. Local analyses will run on the files within your workspace. Whether you have run a local analysis or you are viewing remote issues from a central analysis, the plug-in will present a list of issues in the Issues view, and will also show issue markers alongside the defect occurrences in the source code editor. To help the analysis run smoothly and report accurate information:

- Check the console view for output while running an analysis. Some non-critical warnings (such as number of missing classes) appear only in the console.

- There may be files added to your workspace since your last build that are not available for analysis. In these cases, when you select an analysis option to run, Coverity Desktop will prompt you with several options in the *Uncaptured Source Files* dialog. Select *Capture Build and Analyze* to complete the analysis.

Note

While Coverity Desktop displays issues reported by Test Advisor through Central Analysis, the plug-in does not support local Test Analysis. This means that you will be able to view and triage test policy violations retrieved from the Coverity Connect server, but not analyze local changes for newly introduced test issues.

Also note that central analysis results may contain issues for any Coverity supported language, but Coverity Desktop for IntelliJ and Android Studio only supports local analysis for Java.

Analysis Configurations. Analysis Configurations are entities that define the analysis scope and set the options for local and central analyses. You must have at least one analysis configuration in order to use the plug-in, and you may also set up additional configurations for using different options and analysis scopes.

Each Analysis Configuration can be associated with a reference stream and snapshot from a central analysis on the Coverity Connect server. It is possible, and recommended, to use the analysis settings that are associated with the reference stream. However, you may also choose to pass additional analysis options, or exclude certain files from analysis through your analysis configuration.

1.2.2. Use cases

The use cases described in this section provide a high-level workflow of how you can use Coverity Desktop within your organization. The use cases are:

- Central analysis
- Local analysis
- Central and local analysis

All of the use cases below use the following user roles/personas to demonstrate a particular workflow:

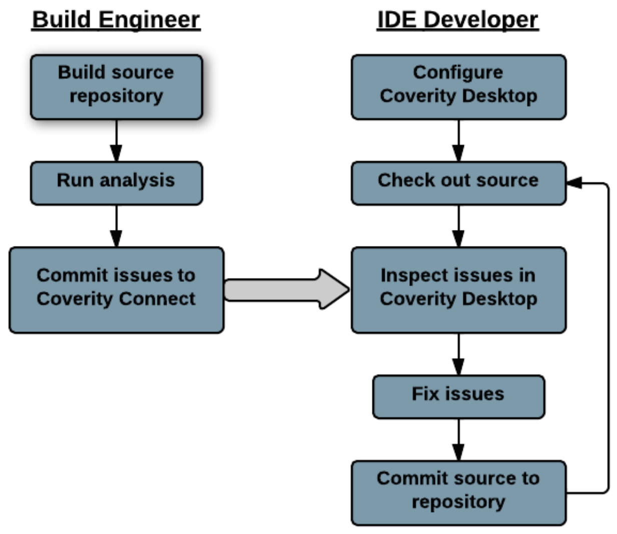
- `Build Engineer` is responsible for the central code repository, including using Coverity tools for Analysis Configuration, analysis runs, and committing the results to a central Coverity Connect server.
- `IDE Developer` is a member of a software development team that uses an IntelliJ based IDE for development and Coverity Desktop to view and triage central analysis and/or local analysis issue results.

1.2.2.1. Use case - Central analysis

Goal. IDE Developer wants to view and triage analysis results from the central source code repository.

The following diagram shows a high-level view of the process:

Figure 1.1. Central analysis model


Build Engineer
IDE Developer
Build source repository
Run analysis
Commit issues to Coverity Connect
Configure Coverity Desktop
Check out source
Inspect issues in Coverity Desktop
Fix issues
Commit source to repository

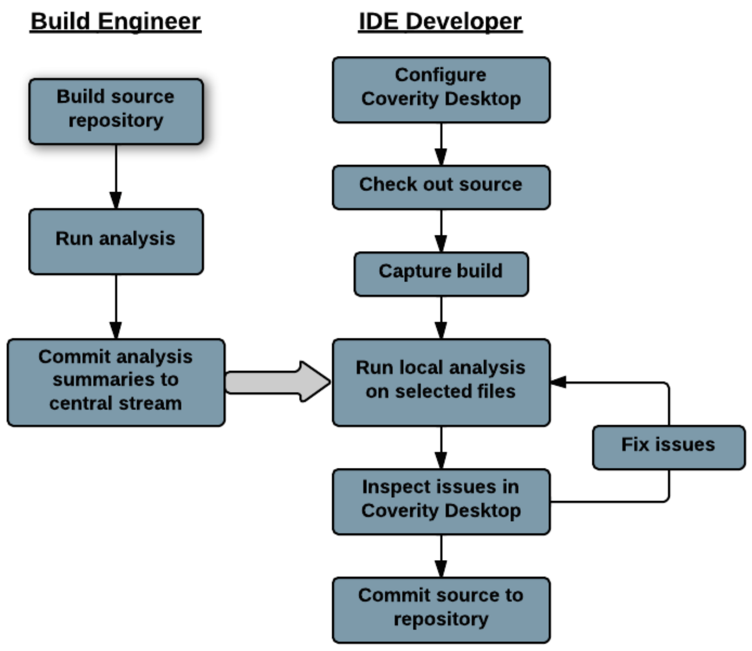
1. Build Engineer builds and runs Coverity Analysis on the central code base. (This process is typically automated on some regular interval, such as nightly).
2. Build Engineer (or automated scripts) commits the issue results to Coverity Connect. (Projects and streams are already appropriately configured to receive issue data).
3. IDE Developer configures Coverity Desktop to connect to the central Coverity Connect server.
4. IDE Developer checks out a section of the code for which he/she is responsible.
5. IDE Developer examines the impact of the remote issues in the code and triages them within the IDE.
6. IDE Developer fixes issues and checks the code into the central repository.
7. The nightly analysis runs.

1.2.2.2. Use case - Local analysis

Goal. IDE Developer wants to analyze and view defect results on his/her local code.

The following diagram shows a high-level view of the process:

Figure 1.2. Local analysis model


Build Engineer
IDE Developer
Build source repository
Run analysis
Commit analysis summaries to central stream
Configure Coverity Desktop
Check out source
Capture build
Run local analysis on selected files
Fix issues
Inspect issues in Coverity Desktop
Commit source to repository

1. Build Engineer builds and runs Coverity Analysis on the central code base. (This process is typically automated on some regular interval, such as nightly).

2. Build Engineer (or automated scripts) commits the issue results, along with function summary data, to Coverity Connect. (Projects and streams are already appropriately configured to receive issue data).

3. IDE Developer configures Coverity Desktop to connect to the central Coverity Connect server and configures local analysis settings.

4. IDE Developer checks out a section of the code for which he/she is responsible.

5. IDE Developer runs local analysis on selected file(s).

6. IDE Developer examines and triages the defects that were found by the analysis.

7. IDE Developer fixes a list of defects.

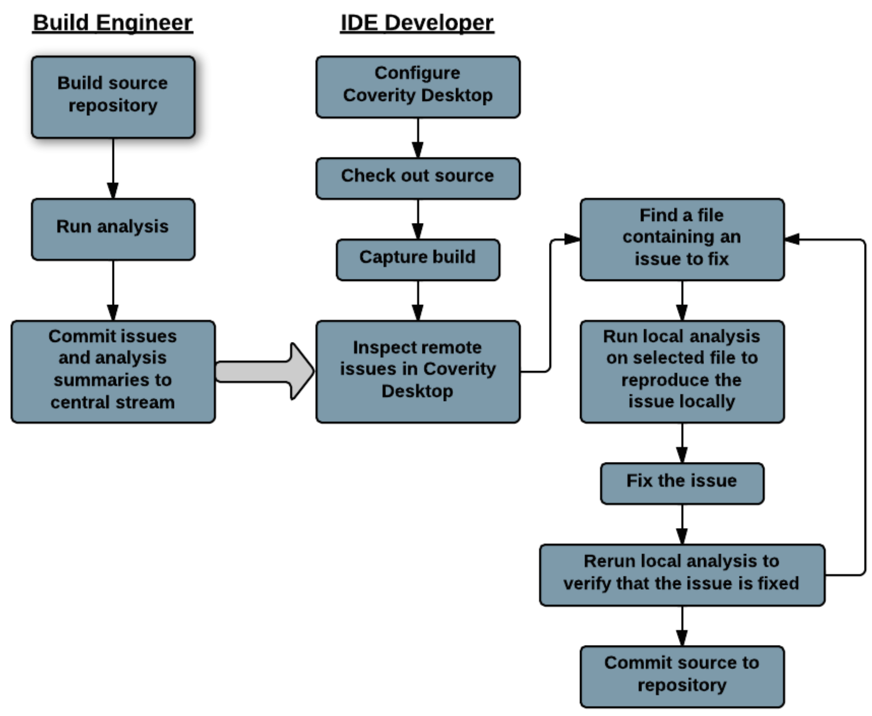
8. IDE Developer runs local analysis again to ensure that the defects were fixed.

9. IDE Developer checks the code into the central repository.

1.2.2.3. Use Case - Using Coverity Desktop with central and local analysis

Goal. IDE Developer wants to see remote issues and also wants to use local analysis before committing code to the source repository. This ensures that the IDE Developer doesn't introduce any new defects into the source repository as a result of their changes.

Figure 1.3. Central and local analysis model


Build Engineer
IDE Developer
Build source repository
Run analysis
Commit issues and analysis summaries to central stream
Configure Coverity Desktop
Check out source
Capture build
Inspect remote issues in Coverity Desktop
Find a file containing an issue to fix
Run local analysis on selected file to reproduce the issue locally
Fix the issue
Rerun local analysis to verify that the issue is fixed
Commit source to repository

1. Build Engineer builds and runs Coverity Analysis on the central code base. (This process is typically automated on some regular interval, such as nightly).
2. Build Engineer (or automated scripts) commits the issue results, along with function summary data, to Coverity Connect. (Projects and streams are already appropriately configured to receive issue data).
3. IDE Developer configures Coverity Desktop to connect to the central Coverity Connect server and configures local analysis settings.
4. IDE Developer checks out a section of the code for which he/she is responsible.
5. IDE Developer retrieves remote issues from the Coverity Connect server, and finds the file containing the defect to be fixed.
6. IDE Developer runs local analysis on the file to reproduce the defect locally.
7. IDE Developer fixes the defect in question.
8. IDE Developer runs local analysis again to verify that the defect is fixed.
9. IDE developer continues working on defects in individual files, or checks the completed code into the central repository.

Chapter 2. Getting Started

Table of Contents

This chapter provides you with the basic steps for retrieving remote issues and running local analyses. If you have not set up an initial analysis configuration, you will be prompted for pertinent configuration information as you perform these tasks for the first time. You can find more detailed information on the various Coverity Desktop views and features in their respective sections of Chapter 3, *Coverity options and views*.

Installation instructions for Coverity Desktop are located in the *Coverity 2020.12 Installation and Deployment Guide*.

2.1. Retrieving remote issues

Once you have completed the initial plug-in configuration, you can retrieve the issues associated with your Coverity Connect project/stream by navigating to Tools → Coverity → View Issues from Coverity Server. Then, use the Issues View and Details View to view and triage any returned issues.

If the Issues View or Details View do not appear initially, navigate to View → Tool Windows, and select *Coverity Issues* or *Coverity Issue Details*, respectively.

Note

The central analysis workflow assumes that you have access to all source code files associated with your chosen central analysis stream. Each of these files should be saved in an IntelliJ or Android Studio project.

2.2. Running a local analysis

Local analysis allows you to locate new defects for code that you have changed before checking it into your central code repository. This short tutorial will guide you through the necessary steps.

In order for local analysis to return the most accurate results, you must connect to a relevant stream on the Coverity Connect server. This will provide necessary interprocedural analysis summary information for fast, accurate analysis results. This will also allow issue triage information to be shared between Coverity Desktop and Coverity Connect. See Chapter 4, *Analysis Configurations Dialog* for additional information.

Note

This tutorial assumes that you have a Java project open within the IDE.

1. Ensure that the source file you want to analyze is open in the main *Editor* pane.

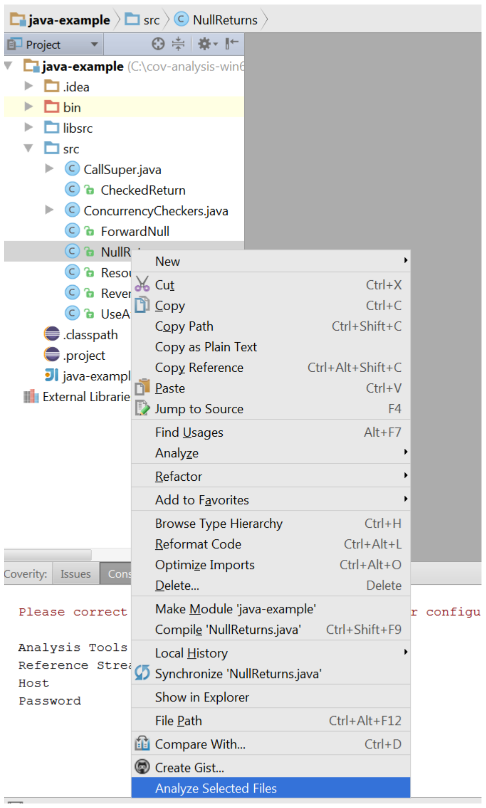
2. Select *Analyze Current Editor File* from the Tools → Coverity menu, or simply click the shortcut icon ( ) on the toolbar.

   Note

   You can also analyze multiple files in the same local analysis. To do so:

   1. Highlight each of the files, packages, and/or projects you wish to analyze in the package explorer.
   2. Right-click the selection to open the context menu.
   3. Click *Analyze Selected File(s)*.

Figure 2.1. Context menu



```
Please correct                r configu

Analysis Tools
Reference Strea
Host
Password
```

Note that if your project uses a gradle build, any source files that have not been previously captured by Coverity Desktop may not be available for analysis. When you attempt to run local analysis on one of these files, you will be prompted with several options in the *Uncaptured Source Files* dialog. To proceed with analysis, select *Capture Build and Analyze*. This will capture a build of any uncaptured files, and then run the local analysis as requested.

3. Once the analysis is complete, the *Issues* view will be updated to reflect any issues in the code you've selected. Double-click on a CID to view and triage the issue from within the IDE.

See Section 3.2.2, “Analysis scope options” for additional analysis options.

Chapter 3. Coverity options and views

Table of Contents

Coverity Desktop allows you to perform a local analysis on your source code from your IDE, so you can quickly pinpoint and fix issues before checking your code into a central code base. This chapter provides details on the various analysis options and views available in the plug-in.

3.1. Toolbar shortcuts

Coverity Desktop adds several shortcuts to the IntelliJ IDEA toolbar, allowing you to quickly select and/or edit analysis configurations, or run a local analysis. The shortcuts are:

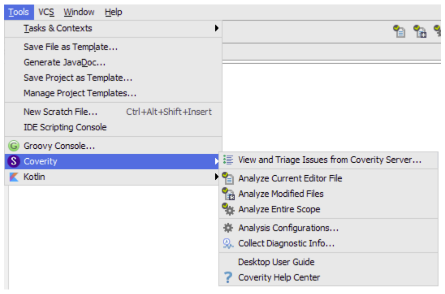
Table 3.1. Toolbar shortcuts

Shortcut	Action	Description
✓	Analyze current editor file	Analyze the current editor file, using the settings specified by the active analysis configuration.
✓	Analyze modified files	Query the configured SCM to see which files have been modified locally, then run local analysis on those files.
✓	Analyze entire scope	Run local analysis on the entire scope of source files included by the active analysis configuration.
Drop-down menu	Active analysis configuration	Drop-down menu that allows you to select the active analysis configuration. You can also create a new configuration, or edit an existing configuration, from this menu.

3.2. Coverity menu

You can configure analysis options, retrieve remote issues, and run local analyses from the Coverity menu.

Figure 3.1. Coverity menu


Tools
VCS
Window
Help
Tasks & Contexts
Save File as Template...
Generate JavaDoc...
Save Project as Template...
Manage Project Templates...
New Scratch File... Ctrl+Alt+Shift+Insert
IDE Scripting Console
Groovy Console...
Coverity
Kotlin
View and Triage Issues from Coverity Server...
Analyze Current Editor File
Analyze Modified Files
Analyze Entire Scope
Analysis Configurations...
Collect Diagnostic Info...
Desktop User Guide
Coverity Help Center

This menu contains the following menu items:

- View Issues from Coverity Server
- Analyze [Current Editor File | Modified Files | Entire Scope]
- Analysis Configurations...
- Collect Diagnostics for Coverity Support
- Coverity Plugin User Guide
- Coverity Help Center

3.2.1. View Issues from Coverity Server

This item will populate the Issues view with relevant issues downloaded from Coverity Connect. These are the most recent issues in the central streams specified in the current Analysis Configuration.

3.2.2. Analysis scope options

There are three ways that Coverity Desktop chooses which files to analyze with Desktop Analysis:

Analyze Current Editor File
: This option will run Desktop Analysis on the currently displayed source file. If there are any unsaved changes, the file will be saved when this option is selected.

Analyze Modified Files (`<SCM>`)
: This option will run Desktop Analysis on all of the source files that are new or modified, relative to the code version most recently checked out from the specified `<SCM>`.

: This option is not available if you have not configured an SCM system. See Section 4.4, “SCM”.

Analyze Entire Scope
: This option will run Desktop Analysis on all of the source files in your active Analysis Configuration's scope.

3.2.3. Analysis Configurations...

This option takes you to the Analysis Configurations Dialog, which contains important options and information for central and local analysis.

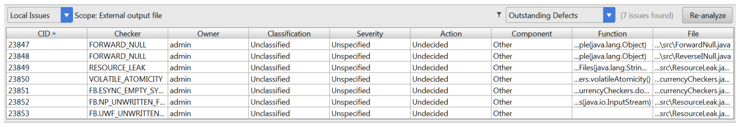
3.2.4. Coverity Desktop Plug-in User Guide and Help Center

These menu items open this user guide and the full Coverity documentation set, respectively. The latter provides additional information about related Coverity components.

3.3. Issues view

The *Issues* view displays in one of two modes:

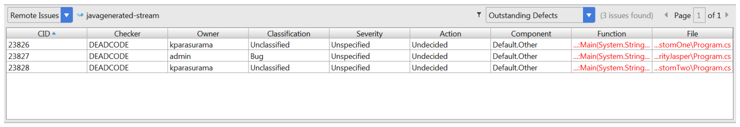
- *Local Issues* mode displays the results of a local analysis.



Local Issues | Scope: External output file | Outstanding Defects | (7 issues found) | Re-analyze

CID	Checker	Owner	Classification	Severity	Action	Component	Function	File
23847	FORWARD_NULL	admin	Unclassified	Unspecified	Undecided	Other	...ple(java.lang.Object)	...\src\ForwardNull.java
23848	FORWARD_NULL	admin	Unclassified	Unspecified	Undecided	Other	...ple(java.lang.Object)	...\src\ReverseINull.java
23849	RESOURCE_LEAK	admin	Unclassified	Unspecified	Undecided	Other	...Files(java.lang.Strin...	...src\ResourceLeak.ja...
23850	VOLATILE_ATOMICITY	admin	Unclassified	Unspecified	Undecided	Other	...ers.volatileAtomicity()	...currencyCheckers.ja...
23851	FB.ESYNC_EMPTY_SY...	admin	Unclassified	Unspecified	Undecided	Other	...urrencyCheckers.do...	...currencyCheckers.ja...
23852	FB.NP_UNWRITTEN_F...	admin	Unclassified	Unspecified	Undecided	Other	...s(java.io.InputStream)	...src\ResourceLeak.ja...
23853	FB.UWF_UNWRITTEN...	admin	Unclassified	Unspecified	Undecided	Other		...src\ResourceLeak.ja...

- *Remote Issues* mode displays issues retrieved from the Coverity Connect server.



Remote Issues | javagenerated-stream | Outstanding Defects | (3 issues found) | Page 1 of 1

CID	Checker	Owner	Classification	Severity	Action	Component	Function	File
23826	DEADCODE	kparasurama	Unclassified	Unspecified	Undecided	Default.Other	...:Main(System.String...	...stomOne\Program.cs
23827	DEADCODE	admin	Bug	Unspecified	Undecided	Default.Other	...:Main(System.String...	...rityJasper\Program.cs
23828	DEADCODE	kparasurama	Unclassified	Unspecified	Undecided	Default.Other	...:Main(System.String...	...stomTwo\Program.cs

If the Issues view is not open, select Coverity → Windows → Issues.

After running a local analysis or retrieving remote issues, the *Issues* view displays a list of the returned issues. It is possible that the list may be too large for you to effectively manage, or you might want to organize your issues in a manner that makes sense for your particular task. For example, you might want to list only the critical and open issues that are assigned to you. You can use filtering and sorting in the *Issues* view to focus the scope of your issues.

Each of the *Issues* view controls is described in Table 3.2, "Issues view buttons".

Table 3.2. Issues view buttons

Button	Name	Action
⟳	Refresh	Retrieves the data for the current page of issues and configuration information from the Coverity Connect server. **Refresh** is not available in *Local Issues* mode.
Re-analyze	Re-run Analysis	Runs a local analysis using the same scope and options as the most recently completed analysis. You can use this to verify that your changes have fixed existing issues, and have not introduced any new defects.

After you select a manageable number of issues to view, you can double click an issue to examine the issue's complexities and impact on your source code. When you open an issue, the source code file that contains the issue opens in the source editor to the location where the issue occurs. Additionally, opening an issue loads the current triage information for the issue in the Details view.

3.3.1. Sorting issues

Central analysis results are listed in the Issues view and can be sorted by any of the default columns. Right-click on a CID, then use *Show Columns* and *Sort By* to display and sort on the following columns:

Impact
: Issue impact as determined by Coverity Connect: High, Medium, Low, or Audit.

  The Impact value is displayed in the first column, and is represented by a colored icon (a blue 'down' arrow for Low impact, a yellow dash for Medium, and a red 'up' arrow for High) - illustrated in Section 3.3, “Issues view”.

CID
: The numeric identifier that groups similar issues that are found in several analysis snapshots.

Checker
: The checker that reported the issue.

Owner
: The user assigned to resolve the issue. You can change the owner that is assigned to the issue in the Details View.

Classification
: Indicates the state of an issue. The classification levels are Unclassified, Pending, False Positive, Intentional, and Bug. You can change the level assignment in the Details View.

Severity
: Indicates an issue's magnitude of potential risk. The severity levels are Unspecified, Major, Moderate, and Minor. You can change the severity level in the Details View.

Action
: Indicates how an issue is to be handled. The categories are Undecided, Fix Required, Fix Submitted, Modeling Required, and Ignore. You can change the action in the Details View.

Fix Target
: The targeted time frame in which the issue should be fixed.

Legacy
: Displays *True* if the CID is a Legacy issue, otherwise *False*.

Ext. Reference
: An identifier (such as an issue number in a different database) specified by your company.

Component
: The name of the component in which the issue was discovered.

Function
: The name of the function that contains the issue. This column is not sortable.

File
: The file path that contains the issue. If the file is not located, the path displays in red text. If red text is displayed, go to the *File Path Mapping* dialog, and make sure that you have the proper definitions

for stripping the remote path prefix and for adding the proper local path to search (if needed). The displayed path is the path known to Coverity Connect, the tooltip will display the local file path if the file is found locally.

Occurrences
: The number of issues that have this CID.

First Detected
: The date of the analysis in which the issue was first detected.

Last Detected
: The date of the analysis in which the issue was last detected.

Last Triaged
: The date of the last analysis in which a change was made to the issue's triage data.

3.3.2. Issue Filters

To focus the scope of your issues in order to produce a more manageable work list, you can construct and apply a filter to one of your analyzed projects. To apply a filter, select one of the following from the *Filter* pull-down menu:

- A pre-defined filter.
- A customized filter that you have created and saved.

After you have selected a filter, the issue list is updated with the matching results.

3.3.2.1. Pre-defined filters

Pre-defined filters represent common search criteria to quickly focus your list of issues. You can add filter expressions, or edit the attributes and values in these filters to customize them. For information about editing pre-defined filters, see custom filters.

Coverity Desktop provides the following pre-defined filters:

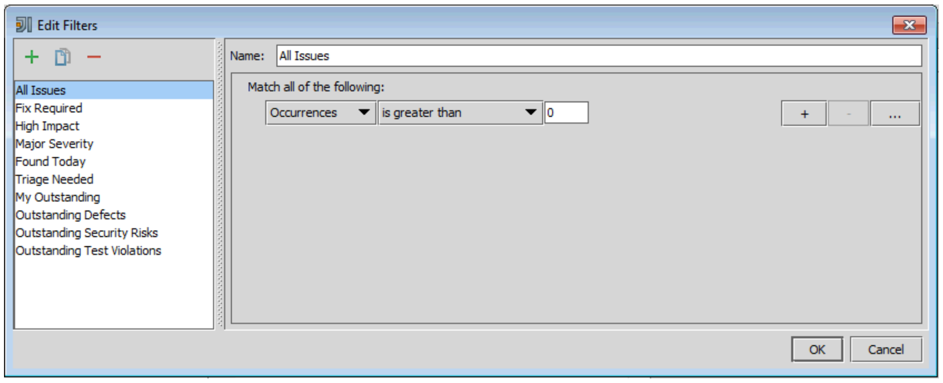
Table 3.3. Built-in filter fields

Column	Description
All Issues	Displays all issues present in the current analysis.
All Outstanding Issues	Displays all outstanding issues present in the current analysis.
Fix Required	Displays all issues that have action - fix required.
Found Today	Displays all issues that were found by the analysis for the current day.
High Impact	Displays all issues that have an impact rating of High.
Major Severity	Displays all issues with a severity of Major.
Triage Needed	Displays all issues that have are unclassified.

Column	Description
My Outstanding	Displays all outstanding issues that are assigned to the current user.
Outstanding Defects	Displays all outstanding quality defects.
Outstanding Security Risks	Displays all security vulnerabilities found by Coverity Analysis.
Outstanding Test Violations	Displays all Test Advisor policy violations.

3.3.2.2. Customized filters

Customized filters allow you to configure one or more filters to restrict the number of issues you wish to be displayed in the *Issues* view. To construct your filter settings, select the *edit filter...* option from the *Filter* pull-down menu. This launches the *Edit Filters* window:



Name: All Issues

Match all of the following:

Occurrences | is greater than | 0

To create a new filter, click the **Add** button (+) and enter the name of the filter in the *Name* field. This is the name that will display in the *Filter* list in the Issues view. If there is a pre-defined or an existing saved filter that you want to be the basis for a new filter, highlight it and select the **Copy** button ( ) to edit the filter's configuration.

Filters are made up of one or more filter expressions. A Filter expression consists of the following:

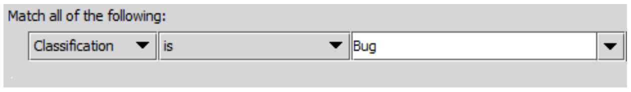
Attribute
: Describes an aspect of the analysis results for the issue. For example, you can specify user-defined triage states, a checker or issue type, and so forth. The attributes that you can define for a filter are described in Table 3.5, "Filter attributes and values".

> **Note**
>
> The attribute and value table does not include any custom attributes that might exist if they were created in Coverity Connect.

Qualifier
: Defines the criteria for how the value, related to its attribute, appears in the search. Each attribute will have a unique list of possible qualifiers from which you can choose, but each qualifier is described in Table 3.6, "Filter qualifiers".

Value
: Represents a specific aspect of an attribute. Attribute values are described in Table 3.5, “Filter attributes and values”.

For example, the following filter expression, when it is saved and applied as a filter, will return only the issues that have a Classification of Bug:



Match all of the following:
Classification | is | Bug

Coverity Desktop allows you to construct more complex filters. The Filter edit buttons allow you to add multiple filter expressions and nested filter expressions. The controls are:

Table 3.4. Filter edit buttons

Button	Action	Description
+	Add	Adds a filter expression at the current level. There are two levels at which you can add a filter, the top level and the nested level. New filter expressions created at the top level are AND operators, so results are returned if they match all of filter expressions that you add at that level. Top-level filter expressions have to contain a unique filter value. The Add button also creates new nested expressions at the same level as the previous nested expression (if one exists).
-	Remove	Removes the current filter expression. If you remove a top level expression that contains nested expressions, all of those expressions are deleted.
...	Add nested	Adds a new nested expression one level under the current top level expression. Nested expressions are OR operators, so results are returned for the current top level expression if the results match the top-level and any of the nested levels. All nested level expressions that belong to the same top level expression must filter on the same attribute. You can change the qualifiers and values.

For example, the following filter expressions, when saved and applied as a filter, return any issues that match the following:

- Any Classification value that is Unclassified

  AND

- Any Impact rating of High Impact

  that has any of the following Action values:

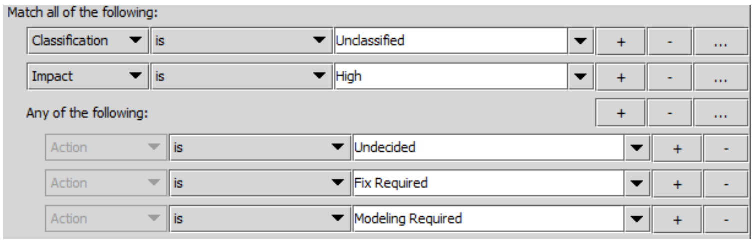
  - Undecided

    OR

  - Fix Required

OR

- Modeling Required



Match all of the following:

Classification is Unclassified

Impact is High

Any of the following:

Action is Undecided

Action is Fix Required

Action is Modeling Required

The following table lists the default attributes and possible values that you can use in a given filter expression. Note that this table does not include customized attributes and values that can be created in Coverity Connect:

Table 3.5. Filter attributes and values

Attribute	**Description**	**Values**
Action	The action to be taken on the issue.	Undecided, Fix Required, Fix Submitted, Modeling Required, or Ignore. For definitions of the action values, see Section 3.4.5.3, “Action”.
CID	The CID (unique numerical representation) of the issue.	A number or range of numbers.
Checker	The name of the checker that reported the issue.	A checker name chosen from a pick list or a full or partial name of a checker entered by the user.
Classification	The classification of the issue.	Unclassified, Pending, False Positive, Intentional, or Bug. Test Advisor specific classifications include Untested, No Test Needed, and Tested Elsewhere. For definitions of the classification values, see Section 3.4.5.1, “Classification”.
Component	Displays issues that are contained in one or more components to which you have access. Component filtering is CID-based, so an issue is included even if only one of its occurrences happens to be in the included component. Components that are invisible to users (through Access Control or exclusion) do not appear in the filter. For more information about	A component name chosen from a pick list (if there is more than one) or a full or partial name of a component entered by the user.

Attribute	Description	Values
	components, see the *Coverity Platform 2020.12 User and Administrator Guide*.	
External Reference	An identifier (such as an issue number in a different database) specified by your company.	An external reference value chosen from a pick list or a full or partial external reference value entered by the user.
File	The name of the file that contains the issue.	A full or partial file name entered by the user.
First Detected	The date of the analysis in which the issue was first detected.	A date or a number of hours, days, weeks, or years.
Fix Target	The targeted time frame in which the issue should be fixed.	Untargeted, or any custom value set by an administrator.
Function	The name of the function that contains the issue.	A full or partial function name entered by the user.
Impact	Impact is a rating of how the issue will affect your code or program. Some issue types have a greater impact on software stability than others. Filtering by impact allows you to view high impact issues first.	High, Medium, Low, or Audit.
Issue Kind	Specifies the type of issue found.	Quality, Security, Test Violation.
Legacy	Specifies whether the CID is a *Legacy* issue.	True or False.
MISRA Category	The classification for MISRA issues	Mandatory, Required, or Advisory (If this choice is left blank, it is set to None.)
Occurrences	The number of issues that have this CID.	A number or range of numbers.
Owner	The owner of the issue. Coverity Desktop gives you a drop-down list of all available users on the Coverity Connect instance to which the plug-in is connected. By default, the owner is the current user.	A Coverity Connect username chosen from a pick list or a full or partial username entered by the user.
Severity	The severity assigned to the issue.	Unknown, Major, Moderate, or Minor. For definitions of the severity categories, see Section 3.4.5.2, “Severity”.

Attribute	Description	Values
Type	Represents a description of the type of issues that one or more checkers might find during analysis.	A checker type chosen from a pick list.

The following table lists the qualifiers that are available to you when you construct your filter expressions. This table is a comprehensive list of the qualifiers. Attributes will only contain a certain subset of these qualifiers (based on the type of attribute):

Table 3.6. Filter qualifiers

Qualifier	Action
is	Includes the value for the attribute in the filter.
is not	Includes all other values (excluding the value that is specified) for the attribute in the filter.
contains	Include the value for the attribute if the value contains the characters entered in the text field.
matches glob	Include the value if it matches any part of the entered glob pattern. For example, if you choose the Checker attribute and add `FB.*NW*`, the filter will return all available checkers that contain "FB." and the characters "NW", such as FB.UWF_UNWRITTEN_FIELD or FB.NP_UNWRITTEN_FIELD.
starts with	Include the value for the attribute if the value starts with the exact characters entered in the text field.
ends with	Include the value for the attribute if the value ends with the exact characters entered in the text field.
is in the range	Select a range of two dates or numbers.
is greater than	A range that is greater than the number you enter.
is less than	A range that is less than the number you enter.
is after	Select a single date. Returns values that occur after the specified date.
if before	Select a single date. Returns values that occur before the specified date.
is today	Returns the values that match the current date.
in the last	Returns the values that match the specified number of hours, days, weeks, or months.
not in the last	Returns the values that do not match the specified number of hours, days, weeks, or months.

3.3.3. Context Menu

When you right-click a CID in the Issues view, the context menu is displayed. From this menu, you can use the following options:

Note

The context menu is unavailable if you are in offline mode or have multiple CIDs selected.

Show Checker Help
Opens the documentation for the checker that found the issue.

Go to
Opens the issue in the Details view.

View in Coverity Connect
Displays the issue in Coverity Connect via your default browser.

Copy
Copies the currently selected row so you can paste it as a text string into an email message, bug report, and so forth.

Set `<attribute>`
Defines an attribute value for Classification, Severity, Action, and so forth. This is a quick way to triage your issue without having to use the Details view. You can select multiple CIDs and set the triage value for all of the selected CIDs at the same time. If a value is greyed out, that means it is an invalid value for the Issue Kind(s) of the selected CIDs.

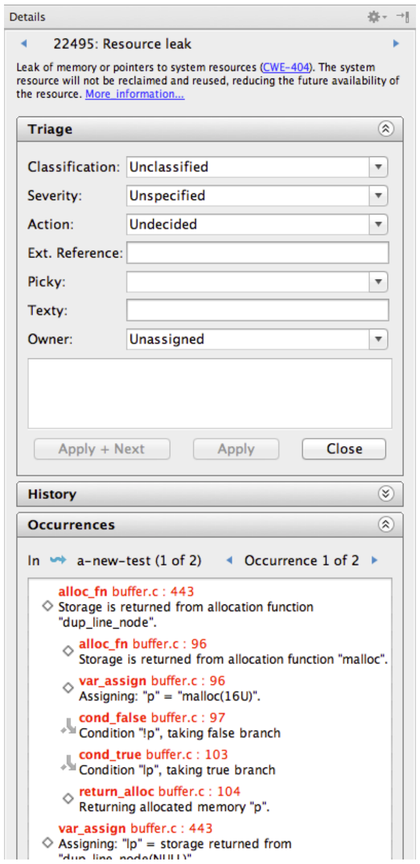
3.4. Details view

The *Details* view allows you to view and triage issue details. For example, you can assign an issue's classification, severity level, owner, or add a comment. The *Details* view is not populated until you double click an issue in the *Issues* view.

Note

The attributes described in this section represent the fields that are available to you by default. It is possible that there are additional attributes and values created as custom attributes in Coverity Connect.

Use the next/previous issue buttons at the top of the *Details* view to display the next or previous CID in your list.



Leak of memory or pointers to system resources (CWE-404). The system resource will not be reclaimed and reused, reducing the future availability of the resource. More information...

Classification: Unclassified
Severity: Unspecified
Action: Undecided
Ext. Reference:
Picky:
Texty:
Owner: Unassigned

In a-new-test (1 of 2) Occurrence 1 of 2

```
alloc_fn buffer.c : 443
Storage is returned from allocation function
"dup_line_node".
    alloc_fn buffer.c : 96
    Storage is returned from allocation function "malloc".
    var_assign buffer.c : 96
    Assigning: "p" = "malloc(16U)".
    cond_false buffer.c : 97
    Condition "!p", taking false branch
    cond_true buffer.c : 103
    Condition "lp", taking true branch
    return_alloc buffer.c : 104
    Returning allocated memory "p".
var_assign buffer.c : 443
Assigning: "lp" = storage returned from
"dup_line_node(NULL)"
```

3.4.1. Issue information

The top of the *Details* view describes the category of the issue discovered by the Coverity Analysis checker and provides a description of the issue. It also provides a link to the relevant SpotBugs documentation (if applicable) and cross reference to the industry standard Common Weakness Enumeration (CWE). CWE is extremely useful for further research on the impact of your issue. It provides detailed descriptions and examples of the issue type.

3.4.2. Triaging an issue

To triage an individual issue, update its various attributes by completing the following steps:

1. Assign the issue an owner by selecting his/her username from the *Owner* pull-down menu.

2. Set the issue's classification, severity, and action to be taken by selecting the appropriate values from their respective pull-down menus.

   Note

   For detailed information on the issue attributes you can triage, see Section 3.4.5, “Issue attributes”.

3. After you have triaged the issue attributes, click **Apply**. Your applied triage state is updated in Coverity Connect for the central analysis stream specified in the active analysis configuration, or for the private stream associated with the local analysis. You can also click **Apply + Next** to apply your changes and move on to the next issue in the list.

The **Close** button closes the issue in the Details view and deselects the open issue in the Issues view. If you have unsubmitted triage changes, Coverity Desktop will prompt you to either apply the changes, close without applying the changes, or cancel the closing process.

3.4.3. Occurrences tab

The *Occurrences* tab shows the stream in which the occurrence of the issue is located, lists the name of the event(s) that lead to the issue for each category, and the filename and line number where this event was detected. Click the event name to go to the event in the source editor.

Coverity Desktop provides tags that highlight the line where the event appears in the code. Tags that are associated with red markers indicate events and messages that are directly related to the issue. Tags that are associated with green markers indicate conditional branches and the programmatic decisions necessary for this issue to occur. When you double click a marker, the source editor focuses on the section of code in which the event occurs. Coverity Desktop provides the following markers:

Table 3.7. Issue markers

Marker	Description
♦	Issue main event.
◇	Issue event.
«◇	Multiple issue events that occur on the same line.
↳	Path event (True branch).
↲	Path event (False branch).

Coverity Analysis can produce issue events that are grouped in multiple sets (sometimes referred to as multi-event issues). For multi-event issues, events are visually separated into different sets and are displayed in an expandable/collapsible tree. Each event set is distinguished by a checker property.

3.4.4. History tab

Each time an attribute is changed or a comment is added, the user responsible for that change is stored in the database, and the listing on the bottom of the issue summary is updated to reflect the modification history. Click the History tab to see the complete history of the modifications.

3.4.5. Issue attributes

You can set the classification, severity, action, and owner for each issue. These issue attributes are intended to tell only part of the story; if an issue is marked as a `bug`, for example, you might also want to document the reason in the designated text area for comments. The issue attributes are useful to restrict the issue list you are viewing. You can also use these states to filter your issues.

3.4.5.1. Classification

Table 3.8, “Classifications” lists descriptions for each of the possible issue classifications, along with the Issue Kind(s) each classification can be attributed to.

Table 3.8. Classifications

Classification	Description	Issue Kind(s)
Unclassified	This attribute is the default when a new result is inserted. It is intended for results that have yet to be viewed by a developer.	Quality, Security, Test Violation
Pending	An issue that should be fixed eventually, but perhaps is not critical enough to fix in the current source code base, or there are other dependencies that prevent it from being fixed at this time.	Quality, Security, Test Violation
False Positive	Results that are not real issues in the code. If these appear to reflect shortcomings or flaws in the analysis engine, report the issue to `software-integrity-support@synopsys.com`.	Quality, Security
Intentional	While the result might be a real bug according to the C/C++ or C# language, it is not a bug in this code because either the code is not important, or the code can never be exercised in a dangerous way in deployment environments.	Quality, Security
Bug	Reflects a determination that the issue found by Coverity analysis is an issue in the code, and is not a False Positive or Intentional.	Quality, Security
Untested	Reflects a determination that a section of the code analyzed by Test Advisor requires that a test be added to the code.	Test Violation
No Test Needed	Reflects a determination that even though the Test Advisor analysis found a section of code that is not covered by a test, you are aware of the violation and that there is an accepted reason that the code is not covered.	Test Violation
Tested Elsewhere	Indicates that a section of code is tested outside of the set of tests that are specified in the Test Advisor analysis process.	Test Violation

3.4.5.2. Severity

Severities for issues describe how critical the issue is, that is, how much damage the issue will potentially cause to your program. Coverity Connect allows you to add, delete, and rename severity issue attribute values. Coverity Desktop displays these customized attributes if they exist in the most current snapshot. For more information, see the *Coverity Connect User Guide*. The default severity attributes are:

Unspecified
: This attribute is the default when a new issue is inserted. It is intended for those results that have not been viewed by a developer.

Major
: Issues that have been confirmed as major reflect a decision that the bug is of the highest level of urgency. It is probable that major severity bugs should be fixed as soon as possible.

Moderate
: Issues that have been confirmed as moderate severity bugs that should be fixed in the near term.

Minor
: Issues that have been confirmed as minor severity bugs are perhaps of lesser priority.

3.4.5.3. Action

Actions describe what should be done about the issue in question. Coverity Connect allows you to add, delete, and rename action issue attribute values. Coverity Desktop displays these customized attributes if they exist in the most current snapshot. For more information, see the *Coverity Connect User Guide*. The default action attributes are:

Undecided
: This attribute is the default when a new issue is inserted. It reflects that no decision about fixing or ignoring has been made.

Fix Required
: The issue is outstanding and requires a fix; such an issue will continue to appear in future commits.

Fix Submitted
: The issue is fixed in the source code, but the fix has not been identified as Fixed through the build, analysis, and commit processes.

Modeling Required
: An investigation is required 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. This can be a form of false positive in which after the modeling is corrected, the analysis will no longer report this issue.

Ignore
: The issue can be ignored. This might be an appropriate action for a bug of minor severity.

3.4.5.4. Owner

To provide or change the owner of the issue, enter the user's name in the text field.

3.4.5.5. Ext. Reference

You can provide an external reference (such as an issue number in a different database).

3.4.5.6. Comment

You can provide a comment for the issue in the designated text area.

3.5. Test Advisor annotations

Coverity Desktop integrates with Test Advisor by allowing you to create and edit test policy files, and view test policy violations along with your other Coverity issues. To view and triage Test Advisor issues, configure the Issues View to display issues *Found by* Central analysis and *Filtered* by Outstanding Test Violations.

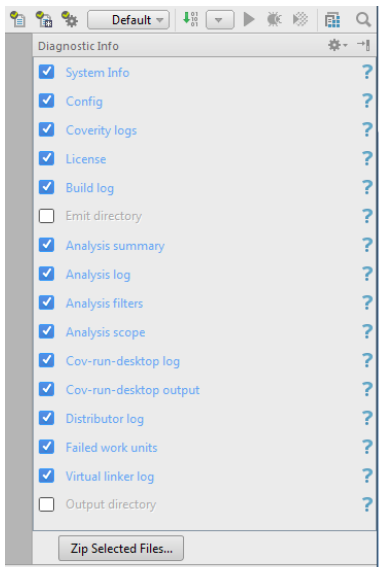
Test Advisor issues, such as coverage, impact, and exclusion data, are represented with various annotations in the left gutter of the text editor pane. See Table 3.9, “Test Advisor annotations” for a description of each. For function level test violations, only coverage data for the relevant function is displayed. For file level test violations, coverage data for the entire file is displayed.

Table 3.9. Test Advisor annotations

Icon	Description
| || |||	**Covered [Old, Med, New]** - Green bars denote that the line or function is covered by one or more tests. The number of bars indicate the code age for the line, with one line for older code, two bars for medium, and three bars for newer code. Hover over the green bar(s) to see a list of tests which cover the line.
⋮ ⋮⋮ ⋮⋮⋮	**Uncovered [Old, Med, New]** - Red bars denote that the line or function is not covered by any test. The number of bars indicate the code age for the line, with one line for older code, two bars for medium, and three bars for newer code.
|	**Coverage Exclusion** - Grey bars denote that the line has been excluded from consideration by Test Advisor. Hover over the grey bar to see the reason for exclusion, if one was specified.
`Month DD,YYYY`	**Impact Date** - Displays the most recent impact date for the line, class, or function. On function and class definition lines, you can hover over the date to see additional impact information or click the link to *View in Coverity Connect.* If the information listed in the pop-up is insufficient, Coverity Connect provides additional data and a richer interface for working with coverage issues. See the *Test Advisor 2020.12 User and Administrator Guide* for more information.

3.6. Collect Diagnostics

This menu option opens the *Coverity Diagnostics* view, which helps you create a zip file of diagnostic information for the current project.



- [✓] System Info
- [✓] Config
- [✓] Coverity logs
- [✓] License
- [✓] Build log
- [ ] Emit directory
- [✓] Analysis summary
- [✓] Analysis log
- [✓] Analysis filters
- [✓] Analysis scope
- [✓] Cov-run-desktop log
- [✓] Cov-run-desktop output
- [✓] Distributor log
- [✓] Failed work units
- [✓] Virtual linker log
- [ ] Output directory

To create the diagnostics zip file:

1. Navigate to Tools → Coverity → Collect Diagnostics for Coverity Support.
2. Click the check boxes to select which diagnostics items you want to include. You can view the information that will be collected for each item by clicking the link, where applicable.
3. Click the **Zip up selected files...** button.
4. Save the zip file to the desired location.

Chapter 4. Analysis Configurations Dialog

Table of Contents

Analysis options and Coverity Connect server information are provided to Coverity Desktop through grouped settings known as analysis configurations. Upon installation of the plug-in, you will have a default analysis configuration in place, which requires some additional settings and connection information to be used for analysis. You may also choose to create additional configurations, to toggle between different options or streams from analysis to analaysis.

The Analysis Configurations Dialog, accessible via the Coverity Desktop toolbar or Tools → Coverity → Analysis Configurations... menu item, allows you to create and edit analysis configurations for Coverity Desktop. To create a new configuration, select *New Configuration* from the *Active Configuration* drop-down, choose a name for the configuration, and then fill out the appropriate information in each of the dialog's subsequent tabs. Optionally, when creating a new configuration, you can copy the settings from an existing configuration (if any) and then edit them in place. The following sections contain details for each of the configuration tabs.

Note

To edit an existing configuration, select its name from the *Active Configuration* drop-down, and update the information in the configuration tabs accordingly.

Additionally, the *Edit Configurations* option (in the *Active Configuration* drop-down) allows you to rename or remove any existing analysis configurations.

Resetting individual fields/values

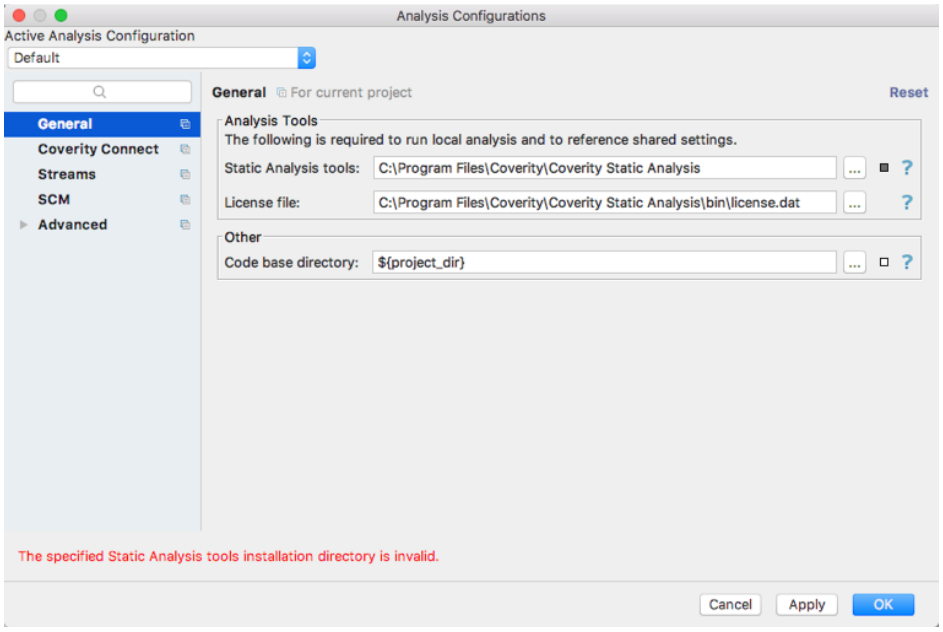
Many of the fields in the *Analysis Configurations* dialog are accompanied by a small grey (or black) "reset box". When you hover over these boxes, they will display how the value was configured (either locally configured, inherited from a `coverity.conf` configuration file, or the Coverity Desktop default value). If the box is black, this means that the value has been set locally. To reset to the original value, click on the box and select *Reset*. This will use the value specified by the `coverity.conf` file, if one exists, or reset to the Coverity Desktop default.

See the *Coverity Desktop Analysis 2020.12: User Guide* for more information on `coverity.conf`.

4.1. General configuration

The *General* tab, as shown in Figure 4.1, “General tab”, specifies the disk locations of your Coverity Analysis tools, including license files, as well as your code base directory. This information is required for running local analysis. This is also required if you want to create a new authentication key file for connecting to Coverity Connect (see Section 4.2, “Coverity Connect” for details).

Figure 4.1. General tab


Analysis Configurations

Active Analysis Configuration: Default

General — For current project — Reset

Analysis Tools

The following is required to run local analysis and to reference shared settings.

Static Analysis tools: C:\Program Files\Coverity\Coverity Static Analysis

License file: C:\Program Files\Coverity\Coverity Static Analysis\bin\license.dat

Other

Code base directory: ${project_dir}

The specified Static Analysis tools installation directory is invalid.

Static Analysis tools
: The full path to your Coverity Analysis installation directory. This is required for local analysis.

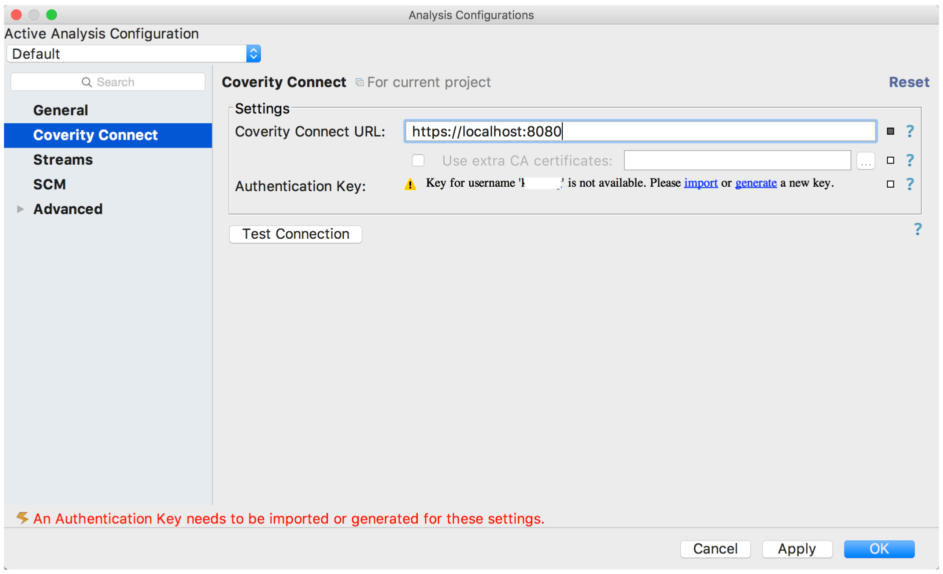
License file
: The location of your `license.dat` file, which contains your existing Coverity Analysis license. The default location is `<install_dir>/bin`.

Code base directory
: The plug-in will store additional Coverity configuration information in the specified directory, as well as look for a `coverity.conf` file here. The `coverity.conf` file may be created by your Coverity Connect or analysis administrator, and used to specify configuration information common to all developers. See the *Coverity Desktop Analysis 2020.12: User Guide* for additional details.

4.2. Coverity Connect

The *Coverity Connect* tab specifies server and authentication information for the Coverity Connect instance relevant to each analysis configuration. This information is required for local analysis, and for retreiving remote issues.

Figure 4.2. Coverity Connect tab



Analysis Configurations

Active Analysis Configuration: Default

Coverity Connect — For current project — Reset

Settings

Coverity Connect URL: https://localhost:8080

Use extra CA certificates:

Authentication Key: Key for username '[redacted]' is not available. Please import or generate a new key.

An Authentication Key needs to be imported or generated for these settings.

Coverity Connect URL

The fully qualified URL for your *Coverity Connect* server. This should include protocol, host name, and port number; for example, `https://connect.synopsys.com:8080`. If you are connecting to a *Coverity Connect* server over SSL, using a certificate signed by a recognized CA, the host name needs to match the name of the host on the certificate. This is not necessary when you use an unsecured connection, or when you use a self-signed certificate from *Coverity Connect*.

Use extra CA certificates

You can also turn on the check box for *Use extra CA certificates* and use the controls to point to a directory that contains additional CA certificates for communicating with Coverity Connect. For additional details, see "Using SSL with Coverity Analysis" in the *Coverity Platform 2020.12 User and Administrator Guide*.

Authentication Key

This field displays information about any existing authentication key file for the specified *Host name* and *Port*.

If authentication fails, or no authentication key exists, you will be prompted to import or generate a new key file. You can generate a new key by clicking the **generate** link and entering your username and password. You can also import an existing key by clicking the **import** link and choosing a key file that's already been created or saved.

If you attempt to import or generate a key when a key file already exists, then a **Replace key** dialog box appears, prompting you to either **Replace** the old key file or to **Save as new**.

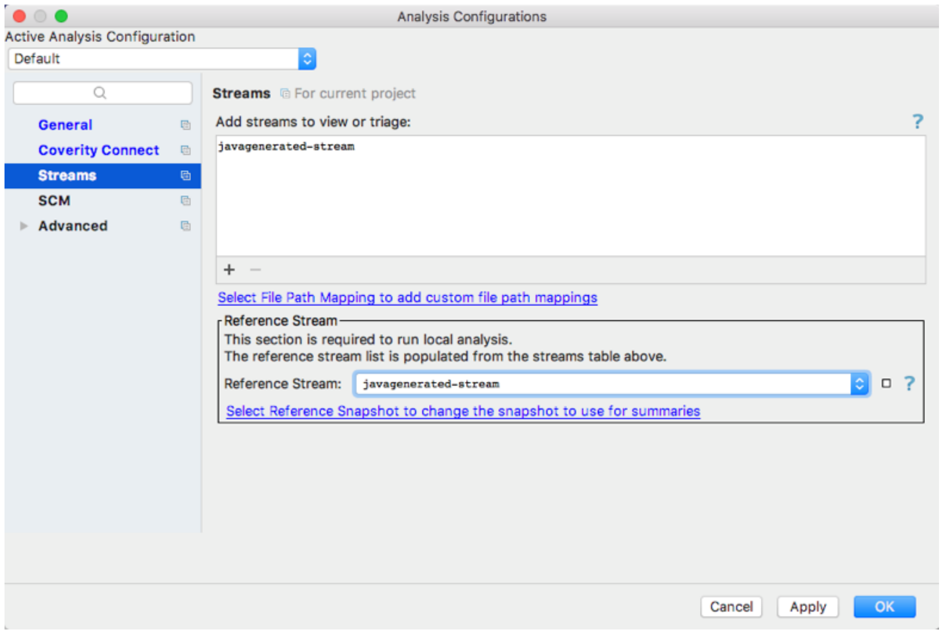
Test Connection

Click this button after entering all connection information. This will test the connection and validate your settings.

4.3. Streams

The *Streams* tab allows you to specify any relevant streams for which you want to view and triage remote issues, as well as select a reference stream and snapshot for local analysis.

Figure 4.3. Streams tab


Analysis Configurations
Active Analysis Configuration
Default
General
Coverity Connect
Streams
SCM
Advanced
Streams
For current project
Add streams to view or triage:
javagenerated-stream
Select File Path Mapping to add custom file path mappings
Reference Stream
This section is required to run local analysis.
The reference stream list is populated from the streams table above.
Reference Stream:
javagenerated-stream
Select Reference Snapshot to change the snapshot to use for summaries
Cancel
Apply
OK

Stream list
: Specifies the streams that contain issues you want to view and/or triage. Click the *Add* (+) button to open the *Select Streams* dialog, which displays a list of all available streams for you to choose from.

  Note that all selected streams should use the same triage store.

Select File Path Mapping...
: This will take you to the File Path Mapping tab.

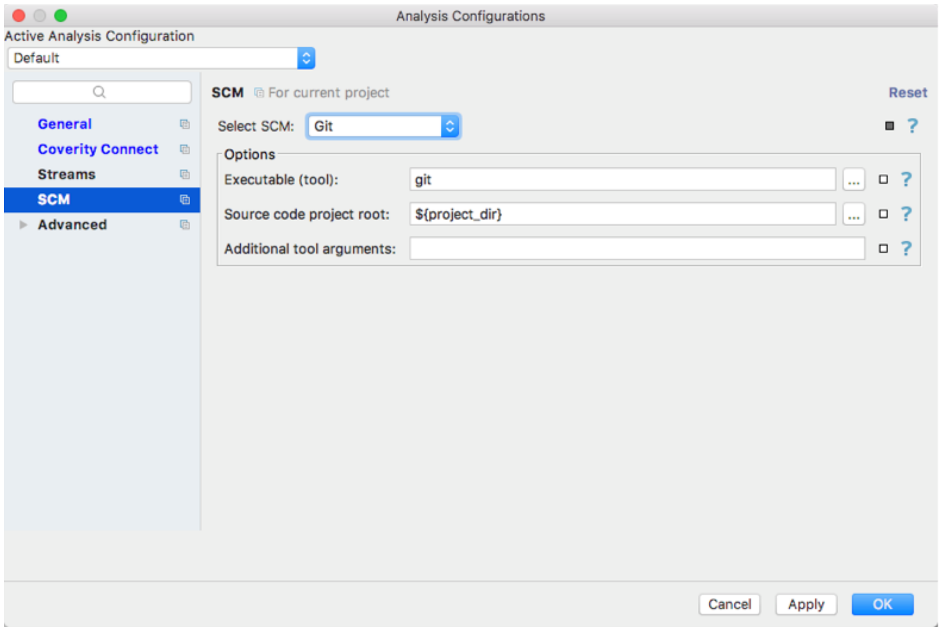
Reference Stream
: Select the reference stream that the analysis configuration will use for local analysis.

Select Reference Snapshot...
: This will take you to the Reference Snapshot tab.

4.4. SCM

The *SCM* tab is an optional configuration, which specifies any SCM information for the current project/ analysis configuration. This is required only if you want to use the Analyze Modified Files command for local analysis.

Figure 4.4. SCM tab



Default

SCM

Select SCM: Git

Executable (tool): git

Source code project root: ${project_dir}

Additional tool arguments:

Select SCM
: Select your source code management system from the drop-down menu. The following SCM systems are currently supported:

  - Accurev
  - Clearcase
  - CVS
  - Git
  - Mercurial
  - Microsoft ADS
  - Microsoft TFS
  - Perforce
  - Plastic
  - Plastic (distributed)
  - Subversion

Executable (tool)
: This is the executable for querying the specified SCM. To be used, the SMC must be installed in the command PATH.

Source code project root
: This is the path to the root of the source repository.

Additional tool arguments
: This is a sequence of additional command line arguments to pass after the executable ("tool") name.

P4PORT (Perforce only)
: The protocol, host name, and port number of the Perforce server, in the format "`<protocol>:<host>:<port>`".

  The `<protocol>` is either `tcp` or `ssl`.

P4CLIENT (Perforce only)
: The name of the Perforce Client, as typically shown in the P4CLIENT environment variable. This is also known as a workspace name.

Remote Repository (Plastic (distributed) only)
: The location of the central Plastic server. This should be in the format of `"<repository>@<remote-server>:<port>"`.

  This field is required.

Authentication Arguments (Plastic (distributed) only)
: If needed, the arguments that will be passed directly to 'cm replicate'. Arguments include, but are not limited to, `--user` and `--password`.

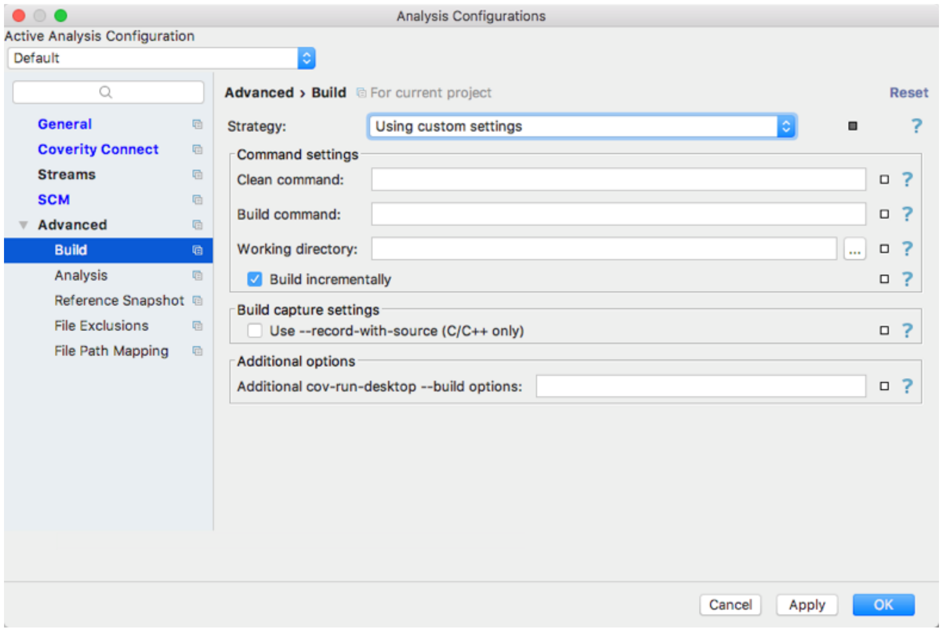
  This field is optional.

4.5. Advanced configuration

The *Advanced* configuration tabs allow you to configure additional build and analysis options for more advanced use cases.

4.5.1. Advanced: Build

Figure 4.5. Advanced: Build tab



Strategy: Using custom settings

Command settings
Clean command:
Build command:
Working directory:
[✓] Build incrementally

Build capture settings
[ ] Use --record-with-source (C/C++ only)

Additional options
Additional cov-run-desktop --build options:

If *Using native builder settings* is selected for *Strategy*, the following options are available:

Additional cov-emit-java options
: Specifies any additional options to be passed to `cov-emit-java`, during the java emit portion of the build.

Gradle Configuration
: Specifies the Gradle configuration which will be built during local analysis.

  This field is only present if using Gradle.

  Note

  Gradle users also need to configure the Coverity Desktop Gradle Plugin. See Chapter 5, *Local analysis with Gradle* for details.

If *Using custom settings* is selected for *Strategy*, the following options are available:

Command settings
: This section contains the options for configuring custom build settings.

  Configure the following settings:

  - **Clean command -** Command line to clean (delete) artifacts produced by the build, so that the next build command will recompile all of the source code. For example, `make clean`.

- **Build command -** Command line to compile the source code. For example, `make` (for C/C++) or `ant` (for Java).
- **Working directory -** The clean and build command will be run from this directory.
- **Build incrementally -** When enabled, selecting the *Capture and Analyze* button in the Uncaptured Source Files dialog will execute the build command and attempt to capture the uncaptured files. The custom Clean command is not executed.

  When running *Capture build of Entire Scope*, this setting is ignored and the custom Clean command is executed.

Build capture settings

- **Use --record-with-source -** When enabled, the build will also capture header file dependencies, which increases the breadth of your analysis, but may also slow down the process considerably.

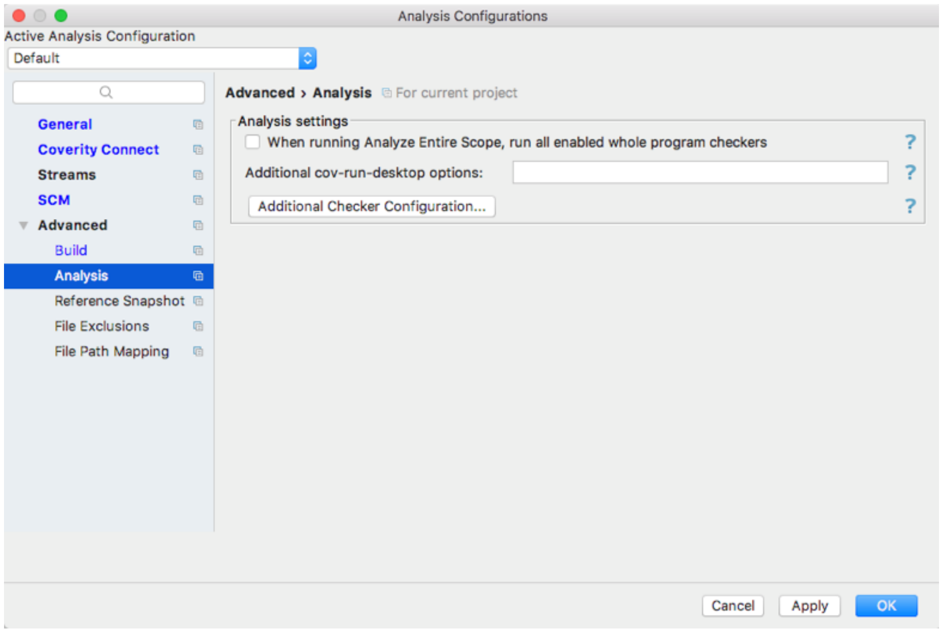
Additional options

- **Additional cov-run-desktop --build options -** Additional options to pass to `cov-run-desktop` during the build. These will be appended to the list above. If there is a conflict, values defined here will be used.

  See the *Coverity 2020.12 Command Reference* for information on available `cov-run-desktop --build` options.

4.5.2. Advanced: Analysis

Figure 4.6. Advanced: Analysis tab



Analysis settings

☐ When running Analyze Entire Scope, run all enabled whole program checkers

Additional cov-run-desktop options:

When running Analyze Entire Scope, run all enabled whole program checkers

If whole program checkers are defined in the *Additional Checker Configuration* dialog, the analysis will attempt to use them. This includes MISRA checkers, web application security checkers, and Android security checkers.

Additional cov-run-desktop options

Specifies any additional options to be passed to `cov-run-desktop` during local analysis.

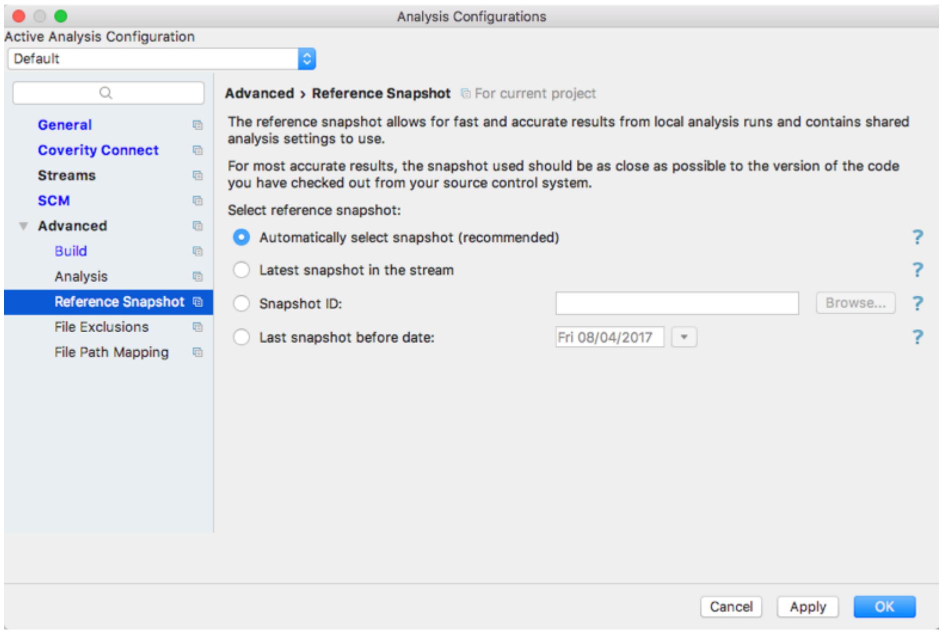
These options will be added to those listed under the *Cov-run-desktop options* field. If there is a conflict, the options specified here take precedent.

Additional Checker Configuration

Two options can be selected here: *Enable checkers that find web application security vulnerabilities* and *Enable checkers that find Android security vulnerabilities*.

4.5.3. Advanced: Reference Snapshot

Figure 4.7. Advanced: Reference Snapshot



The reference snapshot allows for fast and accurate results from local analysis runs and contains shared analysis settings to use.

For most accurate results, the snapshot used should be as close as possible to the version of the code you have checked out from your source control system.

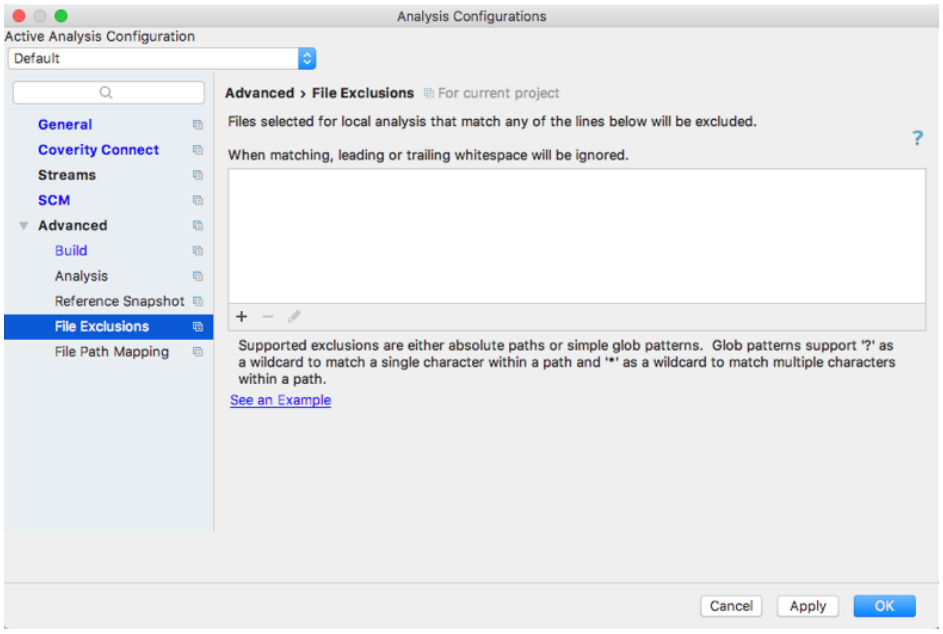
Select reference snapshot:

[✓ Automatically select snapshot (recommended)]
Latest snapshot in the stream
Snapshot ID:
Last snapshot before date: Fri 08/04/2017

The *Reference Snapshot* tab lets you select which reference snapshot to use for local analysis. It is recommended that you use the *Automatically select snapshot* option. This should provide the most accurate analysis results, by using the snapshot closest to the version of the code being analyzed. Alternatively, you can choose to use the latest snapshot in the reference stream, the last snapshot committed before a certain date, or choose a specific snapshot ID.

4.5.4. Advanced: File Exclusions

Figure 4.8. Advanced: File Exclusions



Files selected for local analysis that match any of the lines below will be excluded.

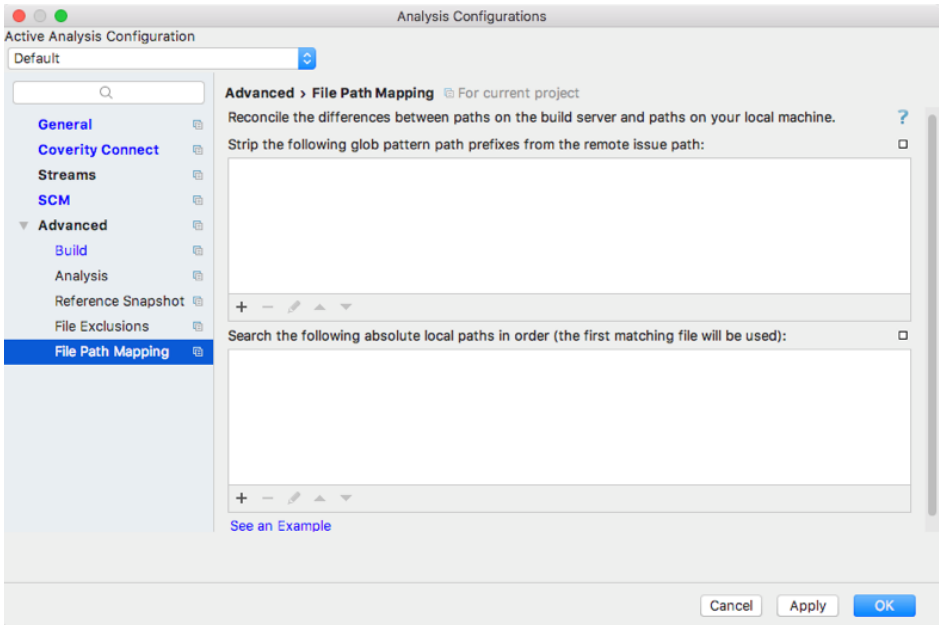
When matching, leading or trailing whitespace will be ignored.

Supported exclusions are either absolute paths or simple glob patterns. Glob patterns support '?' as a wildcard to match a single character within a path and '*' as a wildcard to match multiple characters within a path.

The *File Exclusions* tab lists any files or filepath patterns that should be excluded from local analysis. To add a file exclusion, click the *Add* (+) button and enter the file path or glob pattern to exclude.

4.5.5. Advanced: File Path Mapping

Figure 4.9. Advanced: File Path Mapping



Analysis Configurations

Active Analysis Configuration: Default

Advanced › File Path Mapping — For current project

Reconcile the differences between paths on the build server and paths on your local machine.

Strip the following glob pattern path prefixes from the remote issue path:

Search the following absolute local paths in order (the first matching file will be used):

The *File Path Mapping* tab allows you to translate between remote and local file paths. With path translation, you can configure Coverity Desktop to strip a specified path prefix, and then search local paths for matching files.

In the event that the plug-in cannot resolve the path mapping or locate a source code file locally, the filename will appear in red in the *Issues* view, and you will not be able to view the issue in the source file.

Chapter 5. Local analysis with Gradle

Table of Contents

Coverity Desktop works out of the box to analyze Gradle builds for IntelliJ and Android Studio 3.6 and newer versions

5.1. Working with Gradle and Android Studio (3.0-3.5)

If you are working on an Android project, your `gradle.build` file will declare the Android Gradle plugin dependency as follows:

```
classpath: 'com.android.tools.build:gradle:<version>'
```

Users who want to analyze Gradle builds using an older version(3.0~3.5) of Android Gradle plugin and Android Studio must use the Coverity Desktop Gradle plugin as part of the build process.

Section 5.2 will guide you through setting up the Coverity Desktop Gradle plugin for use with Coverity Desktop for older Android Studio versions.

5.2. Adding the Gradle plugin to your build

To incorporate the Coverity Desktop Gradle plugin, you will have to add some fields to your `build.gradle` file. Make these changes in the top-level build file. This way, you can add configuration options common to all subprojects and modules.

You have two options:

- Reference the maven repository containing the Gradle plug-in, as shown in this code sample.

```
repositories {
    maven{
        url '<cov_connect_host>/coverity-fast-desktop-intellij/maven_repository'
    }
}
dependencies {
    classpath 'com.coverity.desktop:gradle.android.build3.plugin:2020.12-SNAPSHOT'
    . . .
}
```

- Reference the downloaded Gradle plug-in file, as shown in this code sample.

```
repositories {
    jcenter()
    url uri: '<path to the dir where the downloaded zip file was unpacked>'
}
```

```
dependencies {
    classpath 'com.coverity.desktop:gradle.android.build3.plugin:2020.12-SNAPSHOT'
    . . .
}
```

... and in subsequent subprojects or modules in the `build.gradle` file add the following:

```
                    apply plugin: 'com.coverity.desktop.gradle'
```

5.3. Updating to Android Studio 3.6 or newer

Users who are upgrading to Android Studio 3.6 or newer version, please remove Coverity Desktop Gradle plugin and related code from all your `gradle.build` files introduced by section 5.2