



Coverity Wizard 2020.12 User Guide

Coverity Wizard is a component of Coverity Analysis
Copyright 2020 Synopsys, Inc. All rights reserved worldwide.

Table of Contents

1. Using Coverity Wizard to Get Started with Coverity Analysis	1
2. Coverity Wizard Overview	2
2.1. Requirements	2
2.2. About the Coverity Wizard tutorial	2
2.3. Navigation	2
3. Running Coverity Wizard	4
3.1. Configuring Coverity Wizard for Coverity Analysis	4
3.2. Configuring Coverity Wizard for Test Advisor - Development Edition	14
4. Using the Test Advisor Policy Editor and Debugger	27
4.1. Creating a new test policy file	27
4.2. Using the text editor	28
4.3. Using the test policy file outline	28
4.4. Debugging your test policy file	30
4.5. Translation Unit Filter reference	31
5. Using the Guided Test Advisor Policy Creation Wizard	33
5.1. Introduction	33
5.2. Code Age Thresholds	34
5.3. Violation Criteria	34
5.4. Filters	35
6. Troubleshooting Coverity Wizard	37
A. Coverity Glossary	38
B. Coverity Legal Notice	50
B.1. Legal Notice	50

Chapter 1. Using Coverity Wizard to Get Started with Coverity Analysis

You can get started with Coverity Analysis, for any Coverity supported language, by using Coverity Wizard, a GUI-based application.

Scope

This guide covers tasks for setting up and running static quality, security, and test analyses in a centralized (server-based) build system.

Audience

The audience for this guide is administrators (including build engineers and tools specialists) and power users who set up and run the Coverity analyses in an integrated build environment. For details, see [Coverity Analysis Roles and Responsibilities](#) in the *Coverity Analysis 2020.12 User and Administrator Guide*.



Note

It is possible to run analyses from the command-line interface:

- To set up and run analyses from the command line, see *Coverity Analysis 2020.12 User and Administrator Guide*.
- To set up and run test analyses, see *Test Advisor 2020.12 User and Administrator Guide*.
- To set up and run Java dynamic analyses, see *Dynamic Analysis 2020.12 Administration Tutorial*.

Chapter 2. Coverity Wizard Overview

Table of Contents

2.1. Requirements	2
2.2. About the Coverity Wizard tutorial	2
2.3. Navigation	2

Coverity Wizard is a utility that allows you to complete the following Coverity Analysis tasks using an intuitive graphical user interface:

- Configure Coverity tools to work with your compiler.
- Build your source code under Coverity build capture.
- Analyze your source code to find potential quality issues, security issues, and/or test issues.
- Monitor your testing process to ensure its adherence to pre-defined test policy specifications.
- Commit the analysis results to the Coverity Connect server which displays the issues and allows you to manage the issues through a web-based interface.

2.1. Requirements

Coverity Wizard is installed as part of Coverity Analysis. Refer to the *Coverity 2020.12 Installation and Deployment Guide* [🔗](#) for a complete list of hardware and software required.

2.2. About the Coverity Wizard tutorial

The tutorial uses the default values for each screen, and a simple code example with some issues to show you how to run an analysis on your own code. Each option in Coverity Wizard has a help button (🔗) that you can click to get additional information on its functionality and configuration.

2.3. Navigation

Each screen of Coverity Wizard leads you through required configuration steps, providing both basic and more advanced options. You can complete each section in order, or you can go back to a previous screen to make changes at any stage of the workflow. The GUI interface has a left-hand navigation pane that provides an indicator for the active screen. The lower right corner provides buttons that link back to the previous screen, or forward to the next screen.

Indicators provide a visual context of your progress.

Table 2.1. Indicator Icons

Icon	Description
✓	Indicates a completed phase of the workflow.

Coverity Wizard Overview

Icon	Description
!	Indicates that there is an error, or that the step is incomplete.

Additionally, if you'd like to view Coverity Wizard's console commands and output, navigate to View → Show Console at any time.

Chapter 3. Running Coverity Wizard

Table of Contents

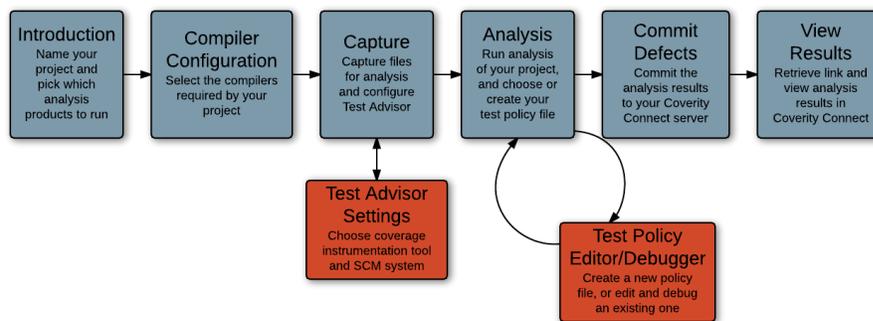
3.1. Configuring Coverity Wizard for Coverity Analysis	4
3.2. Configuring Coverity Wizard for Test Advisor - Development Edition	14

These tutorials run through two examples using Coverity Wizard to set up configuration information.

- The first scenario uses Coverity Wizard to configure Coverity Analysis
- The second scenario illustrates a Test Advisor - Development Edition configuration.

The standard workflow for Coverity Wizard can be seen in Figure 3.1, “Standard workflow”. The red sections denote steps specific to Test Advisor configuration.

Figure 3.1. Standard workflow



3.1. Configuring Coverity Wizard for Coverity Analysis

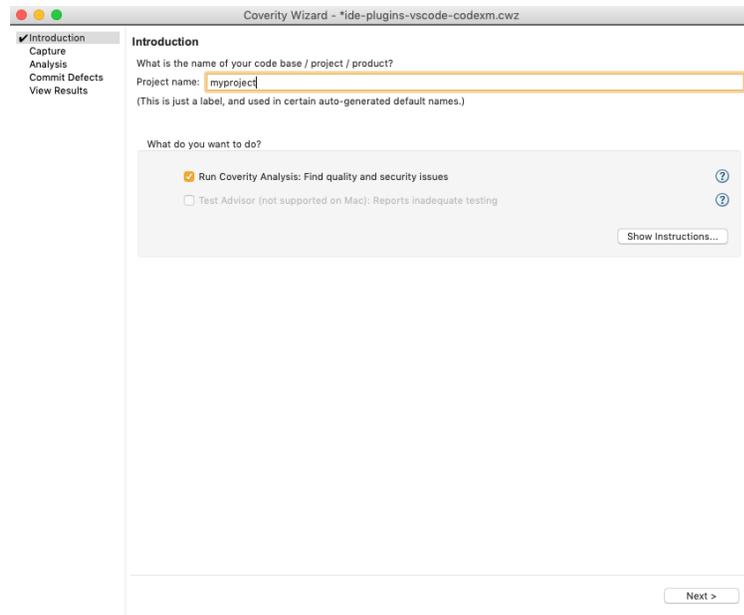
The following sections explain how to get started with Coverity Wizard and how to configure Coverity Analysis projects.

3.1.1. Introduction screen

The Introduction screen is the first of six screens displayed by Coverity Wizard. To launch Coverity Wizard do one of the following:

- On the Windows platform, a shortcut is placed on the desktop if you choose, or you can run the `cov-wizard` command from the `<install_dir>/bin` directory, or have it in your PATH.
- On Linux run the `cov-wizard` command from the `<install_dir>/bin` directory, or have it in your PATH.

Figure 3.2. Introduction screen

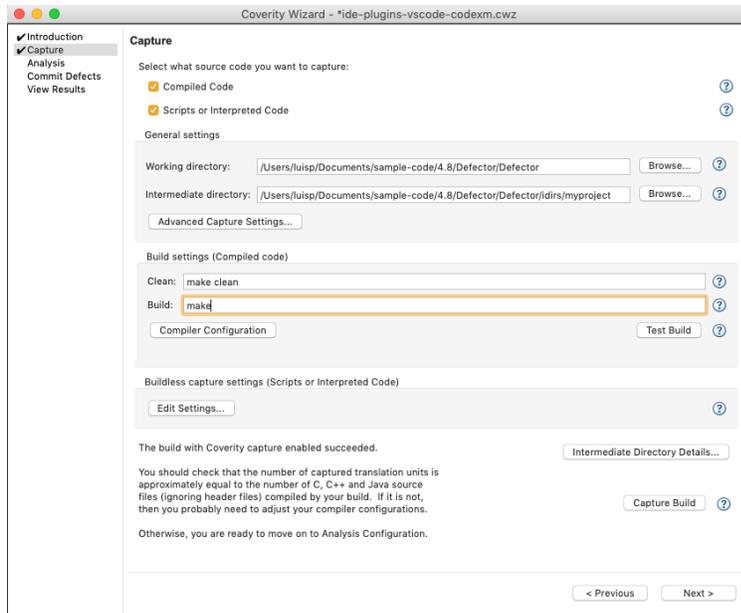


On the Introduction screen, you must choose which products you want to configure (Coverity Analysis, Test Advisor - Development Edition) and select a project name. Click to select *Quality* and, optionally, *Security*; these options indicate the types of analysis you want done. The tutorial project name is `myproject`, however you can customize this name.

3.1.2. Capture

Coverity Wizard provides the ability to configure a command line build and/or buildless capture, and Coverity Wizard will display the results. On Windows you can also have your Integrated Development Environment (IDE) perform the build; see Section 3.1.2.2, "IDE shortcut build".

Figure 3.3. Capture build



Configure the following fields, then click **Capture Build**:

Compiled Code

Select this check-box if your project contains any code that requires compilation. This will run a build capture that will compile and emit relevant files for analysis.

Compiler configuration options are described in Figure 3.4, “Automatically configure compilers”.

Scripts or Interpreted Code

Select this check-box if your project contains any scripts or interpreted code. This will run buildless capture, which will emit relevant files for analysis.

Working directory

For projects with compiled code, the Clean and Build commands will be run from this directory.

Advanced Capture Settings...

This button allows you to set additional `cov-build` options, or configure other advanced settings that relate to the build capture process. Use the help icon (🔗) associated with each option for details.

Intermediate directory

Specifies the directory where all Coverity build and buildless capture information will be stored. This should be on fast local storage to ensure the analysis is as fast as possible.

Build settings (Compiled Code)

It is recommended that you select the **Command line build** option, and specify the **Clean** and **Build** command for any compiled code in your project.

Running Coverity Wizard

You can also choose to have your IDE complete the build capture step. See Section 3.1.2.2, “IDE shortcut build” for more information.

Compiler configuration

This button allows you to configure the compilers you would like to use.

Buildless capture settings (Scripts or Interpreted Code)

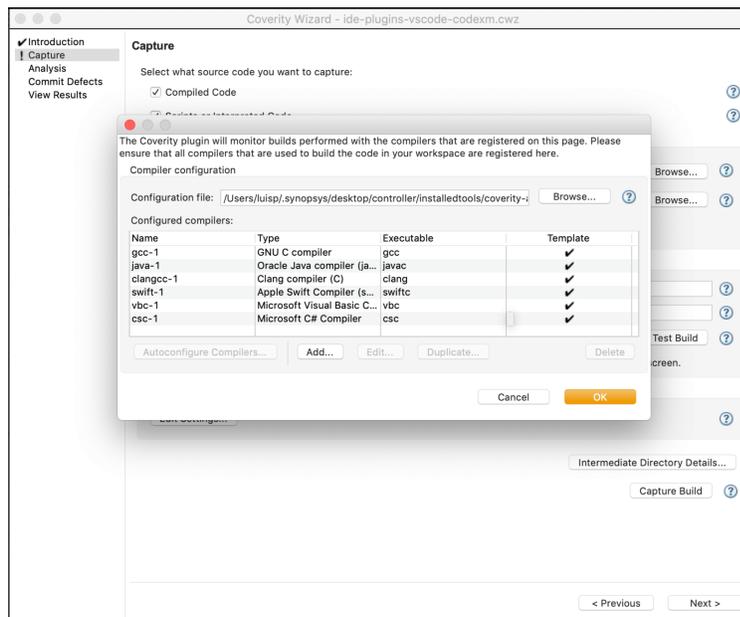
It is recommended that you select the **Scripts** or **Interpreted Code** option to enable the buildless capture feature.

Edit Settings

This button allows you to specify files or directories you would like to include or exclude from the buildless capture.

3.1.2.1. Compiler Configuration

Figure 3.4. Automatically configure compilers



If your project uses any of the compilers listed below, you will only need to click the **Yes** button when prompted to automatically configure the most common compilers.

- GNU C/C++ compiler (gcc)
- Sun/Oracle compiler (javac)
- Clang compiler (clangcc)
- Microsoft Visual C/C++ compiler (cl)
- Microsoft Visual C# Compiler (csc)

Running Coverity Wizard

- Microsoft Visual Basic Compiler (vbc)

Note

If your system uses .NET to compile Visual C# or Visual Basic, `cov-configure` correctly sets up that environment.

The Microsoft Visual C/C++ and Microsoft C# compilers are supported on Windows only. On macOS, Coverity supports only the Unity Roslyn compiler.

If you need to configure another compiler, say for example, the Wind River Diab compiler, follow these steps:

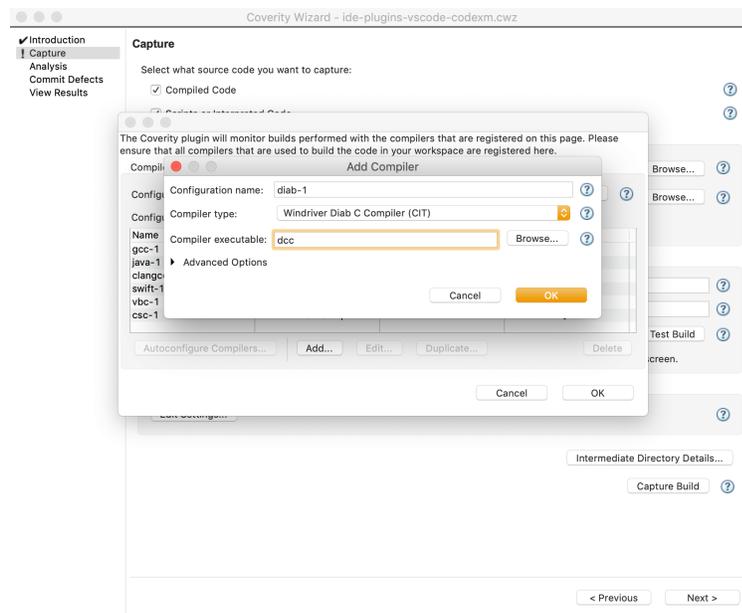
1. Click the **Add...** button.
2. Choose *Windriver Diab C Compiler (CIT)* from the *Compiler type* dropdown menu.
3. Change the configuration name to *diab-1*. You can name it what you want to.
4. Enter the compiler's command line name. For the Wind River Diab C compiler, it is `dcc`.

Note

For a compiler like the Wind River Diab C compiler that has both C and C++ variants, just enter the C compiler's name, and the C++ compiler will be configured automatically.

5. Click **OK** to add it to the list of configured compilers.

Figure 3.5. Add Compiler



3.1.2.2. IDE shortcut build

The **IDE** option allows you to specify a Windows shortcut to your Integrated Development Environment (IDE) to perform the build.

To configure the build settings for use with the Coverity Wizard IDE feature, do the following:

1. Select the **IDE shortcut** button.
2. Set your working directory.



Note

The working directory is where the build commands are run. For some IDEs, for example, Visual Studio, it does not matter what directory you choose to run the IDE from. You can set the working directory to `c:\`.

3. Use one of the options below to activate the IDE option:
 - Enter the filename of the link (.lnk) to your IDE or executable (.exe) to your IDE
 - Use the Browse button to find and choose it.
 - Drag and drop the Windows shortcut from your desktop to the Coverity Wizard interface.
4. Click the **Test Build** button. Your IDE will open if successful. Close the IDE.
5. Click the **Capture Build** button.
6. When your IDE opens, use it to perform a clean and a full build of your code. A pop-up window will remain open reminding you to close the IDE after you perform the clean and full build.
7. Close your IDE.

3.1.3. Analysis Settings

Figure 3.6. Analysis Settings Details

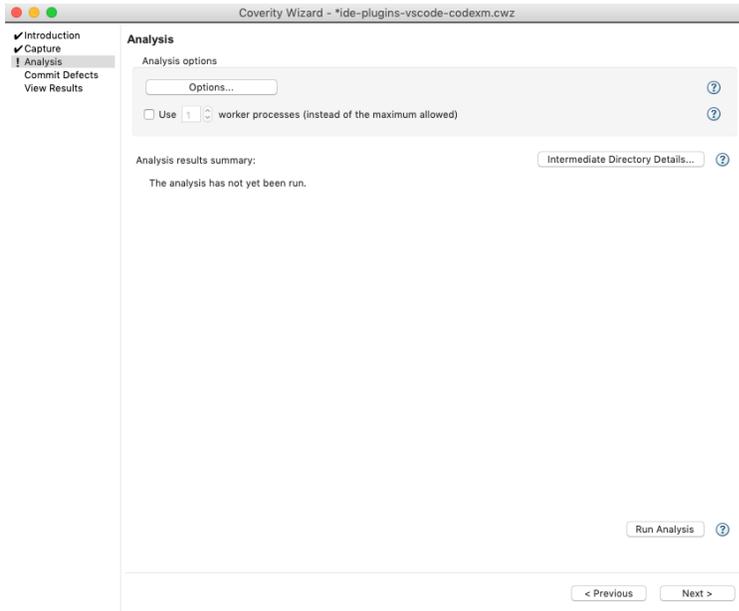


Figure 3.6, “Analysis Settings Details” shows the analysis settings screen which lets you set various analysis options prior to running the analysis.

To set the analysis do the following:

1. Click on the **Run Analysis** button.

The results are displayed in the *Analysis results summary* pane. The *Analysis results summary* pane displays the date and the issues found.

 **Note**

This will perform the analysis with the default analysis settings enabled. To create a custom analysis, click on the **Options** button to access various analysis options prior to running the analysis. Click on the help icons for information on each of the options.

2. Click the **Intermediate Directory Details** button. Expand the Analysis → Defect occurrences button tree node to view defect counts broken down by checker.

3.1.4. Commit Defects

Figure 3.7. Commit Settings Screen

Commit Defects

Coverity Connect settings

Coverity Connect URL: ?

Use extra CA certificates Browse... ?

Authentication Key File: Browse... ?

⚠ No authentication key path entered. Please [import](#) or [generate](#) a new key. ?

?

Stream to commit to

Commit to stream: ? ?

Note: Some pre-existing streams might not be shown. [Why?](#)

Snapshot description: ?

?

?

After analyzing your code, you commit the analysis results to Coverity Connect. This tutorial assumes that you have access to an installed and configured instance of Coverity Connect. You must also have valid user credentials in the form of an authentication key file to connect to Coverity Connect, and the user must have a role with appropriate permissions to commit and view the issues (such as the admin role). If you do not have an authentication key file, you can generate one using your username and password. For installation instructions, see the *Coverity 2020.12 Installation and Deployment Guide*. For more information about user roles, see the *Coverity Platform 2020.12 User and Administrator Guide*.

To commit the analysis results, do the following:

1. Enter the following information in the *Commit Defects* panel:
 - **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*.

Running Coverity Wizard

- *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*.
- For the *Authentication Key File*, click *Browse* to select an existing file.
- If you do not have an authentication key file, click *Generate a new key* to create one. See *Generating a new Authentication Key File*
- Click **Test Connection**.

The *Streams to commit to* panel allows you to either select an existing stream name from the drop down menu or allows you to create a new stream name. If you choose **New**, then a new project and stream will be automatically created with the same name in Coverity Connect.

Note

Coverity Wizard requires streams to have a language set to 'ANY', otherwise they will not appear in the streams drop-down list.

2. For purposes of this tutorial, a new stream named `myprojectxx-cpp` is created, but you can use any name.
3. Click on the **New** button and enter the name of your choice in the *Name* input field. By default, this stream will be available for use with Coverity Desktop Analysis tools. To disable this, please deselect the *Enable Desktop Analysis* checkbox.
4. Click **OK** to finish.
5. Click on the **Commit Defects** button to commit your code to the Coverity Connect.

Note

If your Coverity Analysis Trust Store [🔗](#) has been configured with your server's certificates, you can commit using SSL. To do so, click on **Advanced Commit Settings** and add `--authenticate-ssl` to *Additional arguments*.

3.1.4.1. Generating a new Authentication Key File

Generate Authentication Key File

An authentication key file contains your username and can be used in place of a password when connecting to Coverity Connect. (Note that if you later change your Coverity Connect password, you do not need to regenerate the authentication key file.)

Host name:

Port: Use SSL

Username:

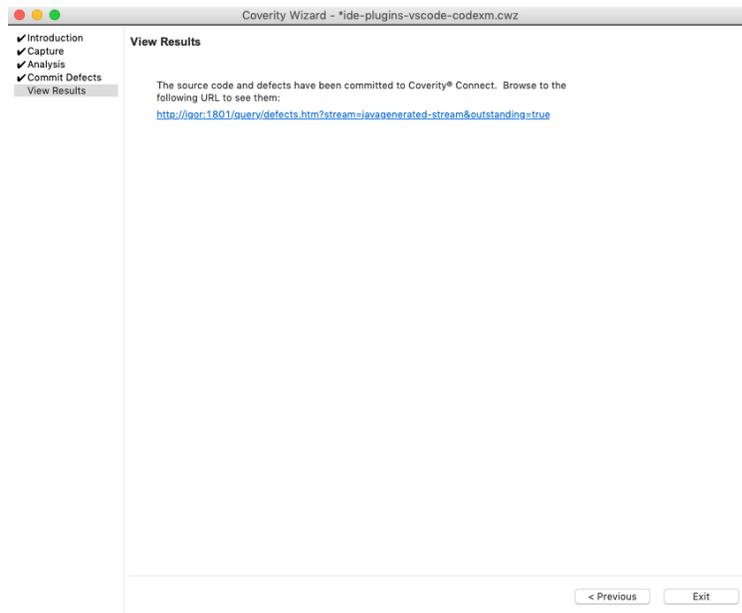
Domain (LDAP only): ?

Password:

In the *Generate Authentication Key File* dialog, enter your *Username* and *Password*. If multiple LDAP servers have been configured for the Coverity Connect server, or if the username exists in both LDAP and Coverity Connect, specify the *Domain*.

3.1.5. View Results

Figure 3.8. Viewing Results Screen

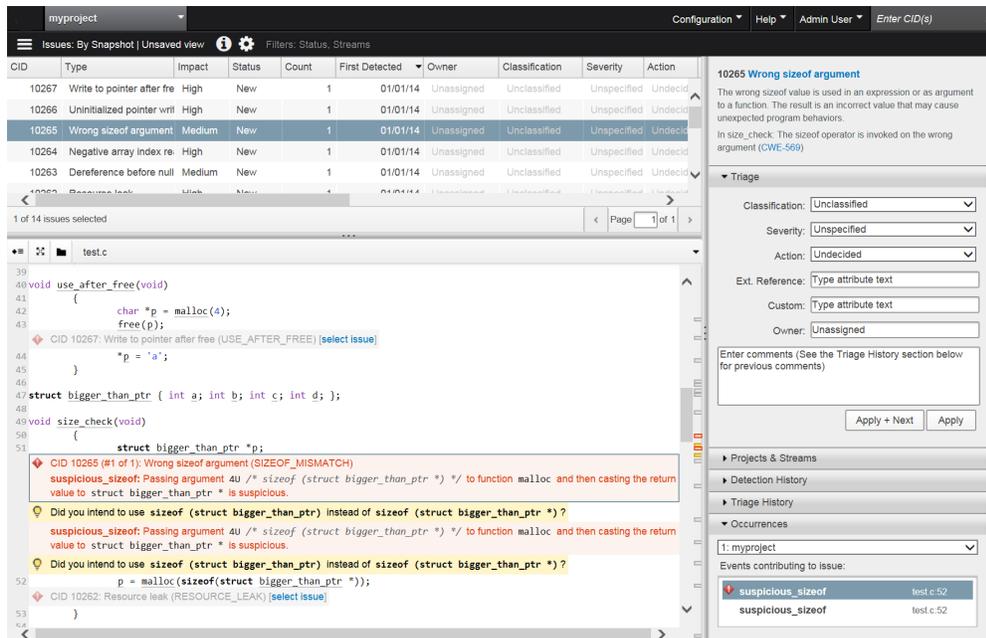


When you select the *Commit Defects* button the analysis results are sent to the Coverity Connect server. A link, as shown in Figure 3.8, “Viewing Results Screen ” takes you to the Coverity Connect application where you can view, manage, and triage the defects. Refer to *Coverity Platform 2020.12 User and Administrator Guide* for detailed information.

You can log into the Coverity Connect application using the same user name and password that you used to commit the defects.

After you log in, you will see the Coverity Connect application as shown in Figure 3.9, “View Defects”

Figure 3.9. View Defects



3.2. Configuring Coverity Wizard for Test Advisor - Development Edition

The following sections provide you with the information necessary to configure Coverity Wizard for Test Advisor - Development Edition projects.

Note

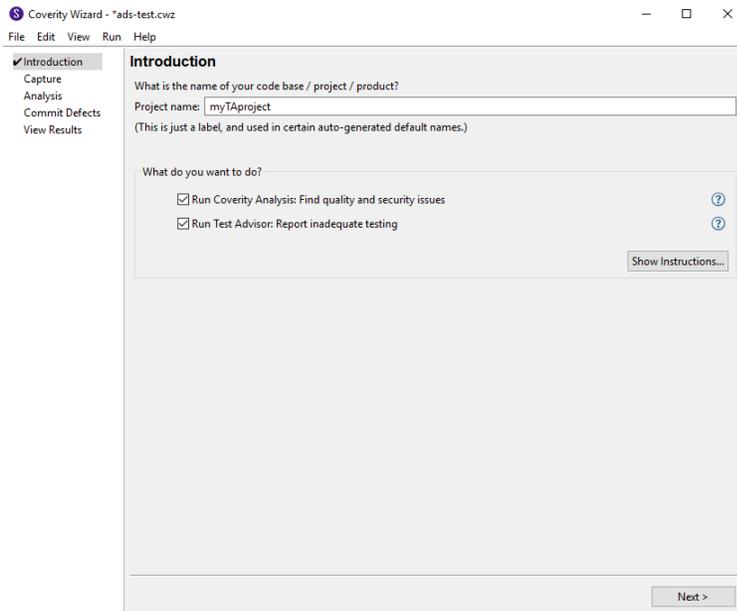
Test Advisor - Development Edition is not supported on Mac OS platforms.

3.2.1. Introduction screen

The Introduction screen is the first of six screens of Coverity Wizard. To launch Coverity Wizard do one of the following:

- On the Windows platform, a shortcut is placed on the desktop if you choose, or you can run the `cov-wizard` command from the `<install_dir>/bin` directory, or have it in your PATH.
- On Linux run the `cov-wizard` command from the `<install_dir>/bin` directory, or have it in your PATH.

Figure 3.10. Introduction screen



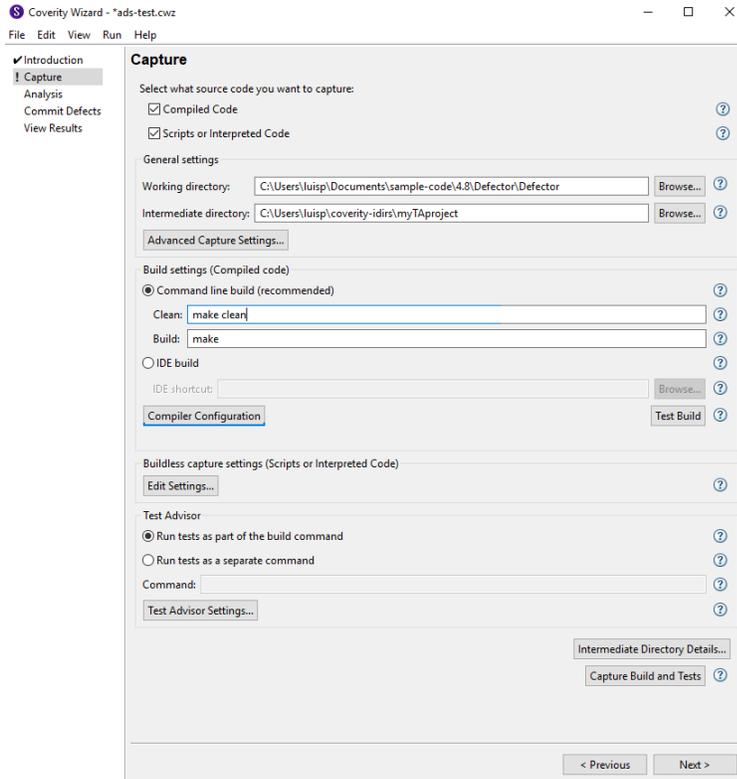
On this screen, you must choose which products you want to configure (Test Advisor - Development Edition in this example) and select a project name. The tutorial project name is `myTApProject`, however you can customize this name.

After you've selected which products you want to use, you can click the **Show Instructions...** button for a brief description of the Coverity Wizard configuration process.

3.2.2. Capture

Coverity Wizard provides the ability to configure a command line build and/or buildless capture, and Coverity Wizard will display the results. On Windows you can also have your Integrated Development Environment (IDE) perform the build; see Section 3.2.2.2, "IDE shortcut build".

Figure 3.11. Capture build



Configure the following fields, then click **Capture Build and Tests**:

Compiled Code

Select this check-box if your project contains any code that requires compilation. This will run a build capture, which will compile and emit relevant files for analysis.

Scripts or Interpreted Code

Select this check-box if your project contains any scripts or interpreted code. This will run a buildless capture, which will emit relevant files for analysis.

Working directory

For projects with compiled code, the Clean and Build commands will be run from this directory.

Advanced Capture Settings...

This button allows you to set additional `cov-build` options, or configure other advanced settings that relate to the build capture process. Use the help icon (🔗) associated with each option for details.

Intermediate directory

Specifies the directory where all Coverity build and buildless capture information will be stored. This should be on fast local storage to ensure the analysis is as fast as possible.

Build settings (Compiled Code)

It is recommended that you select the **Command line build** option, and specify the **Clean** and **Build** command for any compiled code in your project.

You can also choose to have your IDE complete the build capture step. See Section 3.2.2.2, “IDE shortcut build” for more information.

Compiler configuration

This button allows you to configure the compilers you would like to use.

Buildless capture settings (Scripts or Interpreted Code)

It is recommended that you select the **Scripts or Interpreted Code** option to enable the buildless capture feature.

Edit Settings

This button allows you to specify files or directories you would like to include or exclude from the buildless capture.

Test Advisor - Development Edition settings

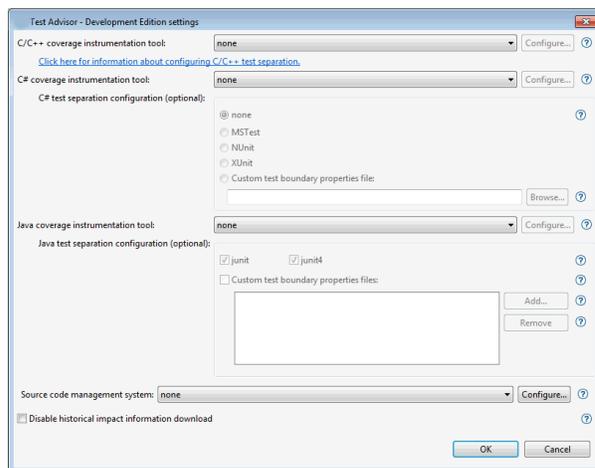
Specify whether to run tests as a part of the build command (default), or to execute a separate command to run the tests after the build.

If you choose to execute an additional command after the build, enter the command in the provided text field.

Test Advisor Settings...

This button will open the *Test Advisor - Development Edition settings* dialog.

Figure 3.12. Test Advisor - Development Edition Settings



Complete the following steps to configure your Test Advisor options:

1. Choose your project's coverage instrumentation tool from the drop-down menus. This example uses the Cobertura coverage tool for Java.

Running Coverity Wizard

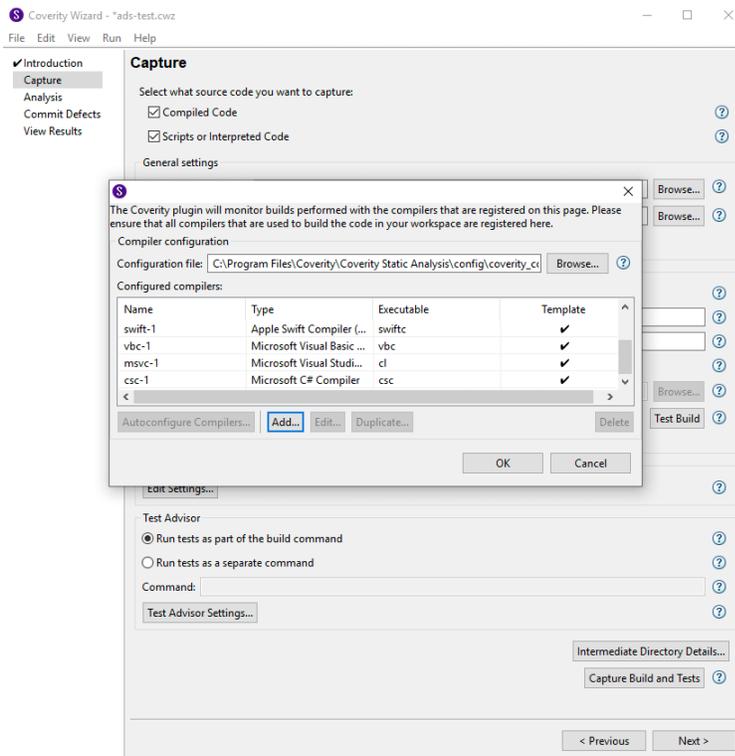
2. If you're running Test Advisor on a Java or C# project, you can also select which test separation configuration you want to use to detect your test boundaries — junit or junit4 for Java; MSTest, NUnit, or XUnit for C#; or you can use a custom test boundary property file for either language. For additional information on setting test boundaries for your Test Advisor projects, see the *Test Advisor 2020.12 User and Administrator Guide* [↗](#).
3. If you would like to include revision histories in your Test Advisor build, choose your project's SCM system from the drop-down menu. Including this information will allow you to create test policy files that take code age and modification dates into account. Some SCMs require additional configuration in the SCM **Configure...** menu.

Note

TFS and ADS are only supported on Windows platforms.

3.2.2.1. Compiler Configuration

Figure 3.13. Automatically configure compilers



If your project uses any of the compilers listed below, then you will only need to click the **Yes** button when prompted to automatically configure the most common compilers.

- GNU C/C++ compiler (gcc)
- Sun/Oracle compiler (javac)

- Clang compiler (clangcc)
- Microsoft Visual C/C++ compiler (cl)
- Microsoft C# Compiler (csc)

 **Note**

The Microsoft Visual C/C++ and Microsoft C# compilers are supported on Windows only.

If you need to configure another compiler, say for example, the Wind River Diab compiler, then follow these steps:

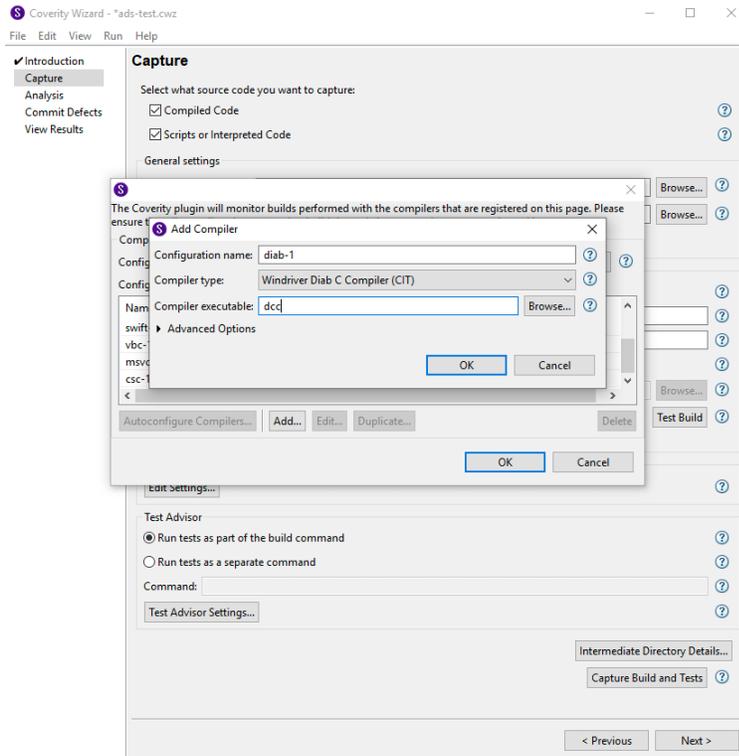
1. Click the **Add...** button.
2. Choose *Windriver Diab C Compiler (CIT)* from the *Compiler type* dropdown menu.
3. Change the configuration name to *diab-1*. You can name it what you want to.
4. Enter the compiler's command line name. For the Wind River Diab C compiler, it is `dcc`.

 **Note**

For a compiler like the Wind River Diab C compiler that has both C and C++ variants, just enter the C compiler's name, and the C++ compiler will be configured automatically.

5. Click **OK** to add it to the list of configured compilers.

Figure 3.14. Add Compiler



3.2.2.2. IDE shortcut build

The IDE option allows you to specify a Windows shortcut to your Integrated Development Environment (IDE) to perform the build.

To configure the build settings for use with the Coverity Wizard IDE feature, do the following:

1. Select the **IDE shortcut** button.
2. Set your working directory.

 **Note**

The working directory is where the build commands are run. For some IDEs, for example, Visual Studio, it does not matter what directory you choose to run the IDE from. You can set the working directory to `c:\.`

3. Use one of the options below to activate the IDE option:
 - Enter the filename of the link (.lnk) to your IDE or executable (.exe) to your IDE
 - Use the Browse button to find and choose it.

- Drag and drop the Windows shortcut from your desktop to the Coverity Wizard interface.
4. Click the **Test Build** button. Your IDE will open if successful. Close the IDE.
 5. Under the *Test Advisor - Development Edition settings* pane, choose whether to run tests as a part of the build command (default), or to execute a separate command to run the tests after the build.

If you choose to execute an additional command after the build, enter the command in the provided text field.

6. Click the **Test Advisor Settings...** button to configure additional Test Advisor options:
 - a. Choose your project's coverage instrumentation tool from the drop-down menus. This example uses the Cobertura coverage tool for Java.
 - b. If you're running Test Advisor on a Java or C# project, you can also select which test separation configuration you want to use to detect your test boundaries — junit or junit4 for Java; MSTest, NUnit, or XUnit for C#; or you can use a custom test boundary property file for either language. For additional information on setting test boundaries for your Test Advisor projects, see the *Test Advisor 2020.12 User and Administrator Guide* [🔗](#).
 - c. If you would like to include revision histories in your Test Advisor build, choose your project's SCM system from the drop-down menu. Including this information will allow you to create test policy files that take code age and modification dates into account. Some SCMs require additional configuration in the SCM **Configure...** menu.

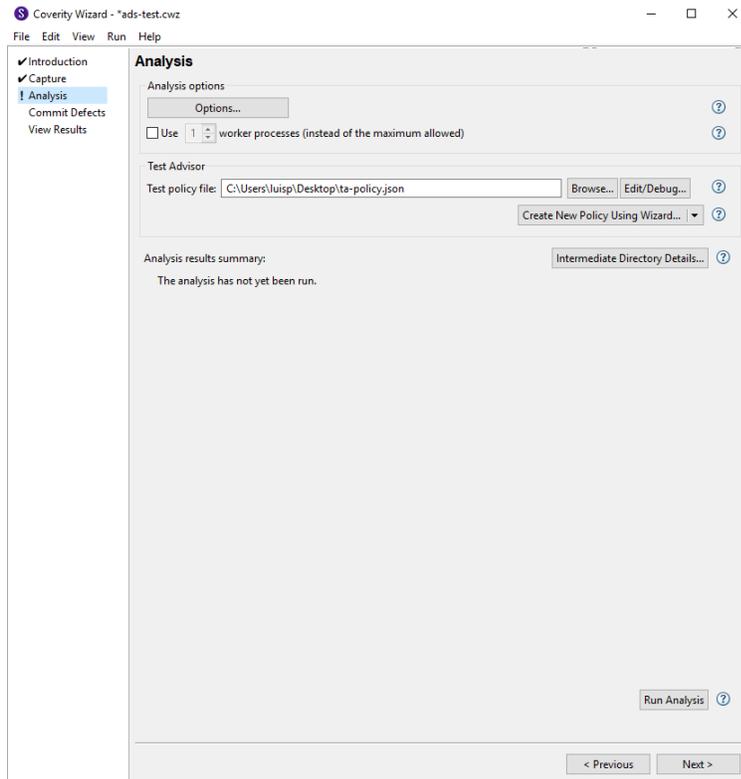
 **Note**

TFS and ADS are only supported on Windows platforms.

7. Click the **Capture Build and Tests** button.
8. When your IDE opens, use it to perform a clean and full build of your code, and run your tests. A pop-up window will remain open reminding you to close the IDE after you perform the clean and full build.
9. Close your IDE.

3.2.3. Analysis Settings

Figure 3.15. Analysis Settings: Test Advisor



To run an analysis on your test policies, complete the following steps:

1. Use the **Browse...** button to navigate to your test policy file.

 **Note**

If you need to create a new test policy file, use the **Create new policy...** drop-down menu to select how you would like to create the file; you can either use the Guided Test Policy Creation Wizard, or start from one of several test policy templates. You can also debug and edit your test policy file directly within Coverity Wizard by clicking the **Edit/Debug** button. This will open the Test Advisor Policy Editor, which has several tools and views to help you create more effective test policy files. See Chapter 4, *Using the Test Advisor Policy Editor and Debugger* for additional details.

2. Click the **Run Analysis** button to complete your test analysis.

The results are displayed in the *Analysis results summary* pane. The *Analysis results summary* pane displays the date and the issues found.



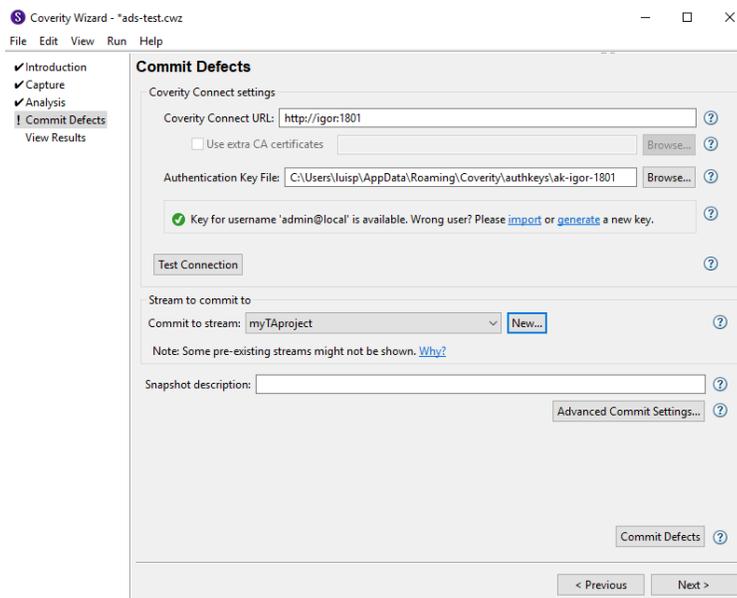
Note

This will perform the analysis with the default analysis settings enabled. If you would like to create a custom analysis, click on the **Options** button to access various analysis options prior to running the analysis. Click on the help icons for information on each of the options.

3. Click the **Intermediate Directory Details** button. Expand the Analysis → Defect occurrences button tree node to view defect counts broken down by checker.

3.2.4. Commit Defects

Figure 3.16. Commit Defects: Test Advisor



In this phase of the workflow, you commit the analysis results to Coverity Connect. This tutorial assumes that you have access to an installed and configured instance of Coverity Connect. You must also have a valid user credential in the form of an authentication key file to connect to Coverity Connect, and you must have a role with appropriate permissions to commit and view the issues (such as the admin role). If you do not have an authentication key file, you can generate one using your username and password. For installation instructions, see the *Coverity 2020.12 Installation and Deployment Guide*. For more information about user roles, see the *Coverity Platform 2020.12 User and Administrator Guide*.

To commit the analysis results, do the following:

1. Enter the following information in the *Coverity Connect settings* panel:
 - *Host Name* of the machine where Coverity Connect is installed.

- Select your connection type (*Secured* or *Unsecured*). A secured connection uses SSL to communicate with the Coverity Connect server, while an unsecured connection uses no encryption.

You can also select the *Add CA certificates* option to point to a certificate or certificate chain file (.pem) for communicating with Coverity Connect. See "Client-side SSL Certificate Management" in the *Coverity Platform 2020.12 User and Administrator Guide* for additional details.

- Enter the port number for the web interface of Coverity Connect. This number is chosen when Coverity Connect is installed. The default port number is 8080 for HTTP and 8443 for HTTPS when using secured connection.
- Type *admin* in the Username field (or another username that is in Coverity Connect that has privileges to commit snapshots).
- Enter the password for the entered user. (The password for the admin user is initially set when Coverity Connect is installed.)
- Click on the **Test Connection** button.

The *Streams to commit to* panel allows you to either select an existing stream name from the drop down menu or allows you to create a new stream name. If you choose **New**, then a new project and stream will be automatically created with the same name in Coverity Connect.

2. For purposes of this tutorial, a new stream named `myprojectxx-cpp` is created, but you can use any name.
3. Click on the **New** button and enter the name of your choice in the *Name* input field. By default, this stream will be available for use with Coverity Desktop Analysis tools. To disable this, please deselect the *Enable Desktop Analysis* checkbox.
4. Click **OK** to finish.
5. Click on the **Commit Defects** button to commit your code to the Coverity Connect.

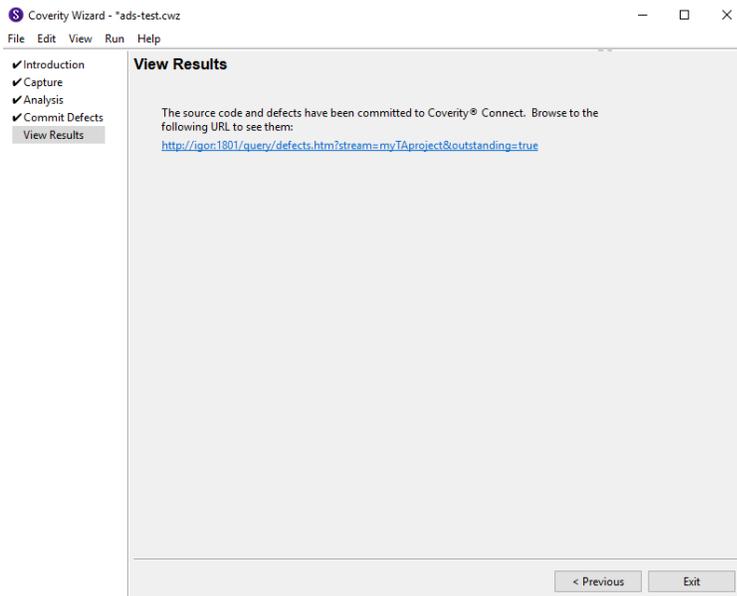


Note

If your Coverity Analysis Trust Store  has been configured with your server's certificates, you can commit using SSL. To do so, click on **Advanced Commit Settings** and add `--authenticate-ssl` to *Additional arguments*.

3.2.5. View Results

Figure 3.17. Test Advisor View Results Screen



When you select the *Commit Defects* button the analysis results are sent to the Coverity Connect server. A link, as shown in Figure 3.8, “Viewing Results Screen ” takes you to the Coverity Connect application where you can view, manage, and triage the defects. Refer to *Coverity Platform 2020.12 User and Administrator Guide* for detailed information.

You can log into the Coverity Connect application using the same user name and password that you used to commit the defects.

After you log in, you will see the Coverity Connect application as shown in Figure 3.18, “View Defects”

Running Coverity Wizard

Figure 3.18. View Defects

The screenshot displays the Coverity IDE interface. At the top, there is a navigation bar with 'myTAproject', 'Configuration', 'Help', 'Admin User', and 'Enter CID(s)'. Below this is a table of issues with columns for CID, Type, Impact, Status, First Detected, Owner, Classification, Severity, Action, and Component. The table lists several 'Insufficient function coverage' issues, with CID 10239 highlighted.

CID	Type	Impact	Status	First Detected	Owner	Classification	Severity	Action	Component
10241	Insufficient function cov	Low	New	05/17/14	Unassigned	Unclassified	Unspecified	Undecided	Other
10240	Insufficient function cov	Low	New	05/17/14	Unassigned	Unclassified	Unspecified	Undecided	Other
10239	Insufficient function cov	Low	New	05/17/14	Unassigned	Unclassified	Unspecified	Undecided	Other
10238	Insufficient function cov	Low	New	05/17/14	Unassigned	Unclassified	Unspecified	Undecided	Other
10237	Insufficient function cov	Low	New	05/17/14	Unassigned	Unclassified	Unspecified	Undecided	Other

The main editor shows the source code for 'LoopingAndConcurrency.java'. The code includes a 'downTheOther()' method with a guarded write and an 'infiniteLoop()' method. Several defects are annotated in the code, including CID 10216 (Unguarded write), CID 10223 (Insufficient function coverage), CID 10231 (Insufficient function coverage), and CID 10239 (Insufficient function coverage). The detailed view for CID 10239 shows a violation of the 1% coverage threshold for the 'infiniteLoop()' function.

On the right side, there is a 'Triage' panel for the selected issue. It includes dropdown menus for 'Classification' (Unclassified), 'Severity' (Unspecified), and 'Action' (Undecided). There are also input fields for 'Ext. Reference', 'Custom', and 'Owner' (Unassigned). Below these are buttons for 'Apply + Next' and 'Apply'. A 'Comments' section is also present.

Chapter 4. Using the Test Advisor Policy Editor and Debugger

Table of Contents

4.1. Creating a new test policy file	27
4.2. Using the text editor	28
4.3. Using the test policy file outline	28
4.4. Debugging your test policy file	30
4.5. Translation Unit Filter reference	31

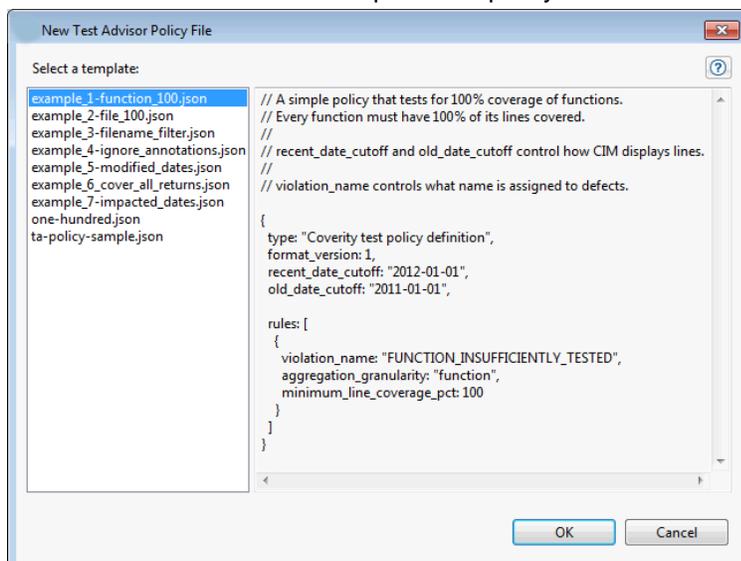
This section provides an overview of the views and options in the Test Advisor policy editor, for use with JSON test policy files. You can access the Policy Editor and Debugger from the Analysis Settings screen in Coverity Wizard. This tool not only allows you to edit and debug your test policy files, but also test the results of the file on your codebase. This will give you the ability to more rapidly update your test policy files and test for your desired results.

You can either edit and debug an existing test policy file by browsing to its location and clicking the **Edit/Debug** button, or create a new policy file by clicking the **Create New Policy Using [Wizard|Template]** button.

This section assumes that you have a working knowledge of Coverity Test Advisor - Development Edition and Test Advisor policy concepts. For more information on Test Advisor and policy files, see *Test Advisor 2020.12 User and Administrator Guide* [\[4\]](#).

4.1. Creating a new test policy file

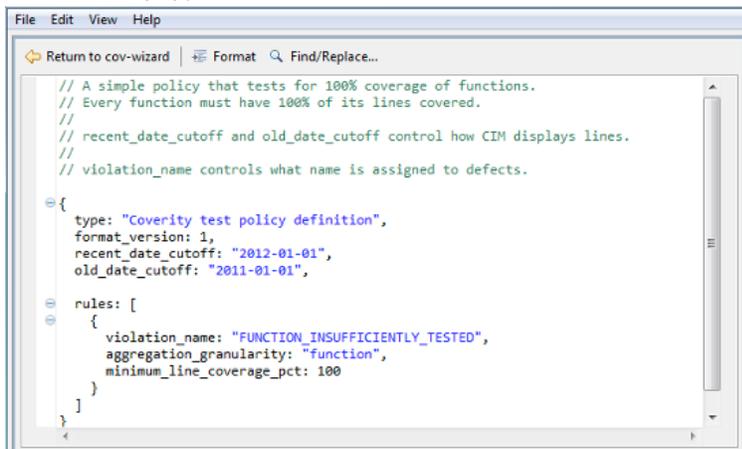
Select **Create New Policy Using Template** to create a new policy file. You will be shown a pop-up window in order to select a template test policy file to start from.



The left pane, titled *Select a template*, lists each of the available template files, while the right pane displays a preview of the file with a short commented description of the test policy file at the top. Select the one that works best for your project and click *OK*.

4.2. Using the text editor

Upon opening the editor, your test policy file is displayed in the main window. You can use this window to examine your test policy file for errors, as well as make changes and save your updates directly within Coverity Wizard. When editing the policy, press *Ctrl-Space* to see a list of context appropriate options for quicker editing/formatting. This will auto-complete the line when only one option exists for what you have already typed.



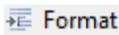
```
File Edit View Help
Return to cov-wizard | Format Find/Replace...

// A simple policy that tests for 100% coverage of functions.
// Every function must have 100% of its lines covered.
//
// recent_date_cutoff and old_date_cutoff control how CIM displays lines.
//
// violation_name controls what name is assigned to defects.

{
  type: "Coverity test policy definition",
  format_version: 1,
  recent_date_cutoff: "2012-01-01",
  old_date_cutoff: "2011-01-01",
  rules: [
    {
      violation_name: "FUNCTION_INSUFFICIENTLY_TESTED",
      aggregation_granularity: "function",
      minimum_line_coverage_pct: 100
    }
  ]
}
```

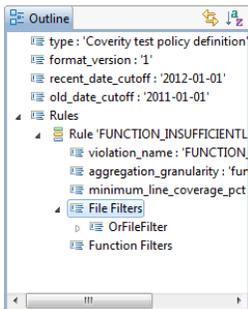
In addition to the full text of your policy file, the text editor features the tools listed in Table 4.1, “Test Advisor Policy Editor tools” to help you create and navigate your test policy file.

Table 4.1. Test Advisor Policy Editor tools

Name	Icon	Description
Format		Reformats the test policy file into clean text-blocks based on bracketed groups.
Find/Replace...		Search and replace tool for text occurrences in your test policy file.
Collapse		Collapse a block of text to display only the top-level element. Click again to expand the section.
Text error		Highlights a text error on a specific line. Hover over the icon for a short description of the issue.
Warning		Highlights a potential text error on a specific line. Hover over the icon for a short description of the issue.

4.3. Using the test policy file outline

The outline view, in the upper-right pane of the editor, displays a basic outline of your test policy file. In this view, you can explore existing sections in the file, and add new elements as needed.



4.3.1. Navigating the test policy file outline

Upon opening the editor, the outline displays each of the top-level elements in the test policy file (*type*, *format_version*, *recent_date_cutoff*, *old_date_cutoff*, *rules*, and *define_filters*). Any element that contains sub-sections can be expanded and collapsed by clicking its corresponding node button ().

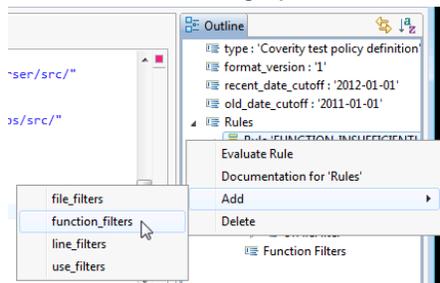
The outline view also provides the tools in Table 4.2, “Test Advisor Policy Editor tools” to help you navigate the file outline.

Table 4.2. Test Advisor Policy Editor tools

Name	Icon	Description
Link with Editor		Links the outline to the file contents in the text editor view. When enabled, clicking on an element in the outline will display and highlight that element in the text editor view. Conversely, the selected item in the outline will also be updated as you click through the text editor.
Sort		Sorts all top-level elements alphabetically. All sub-elements are also sorted within their respective contexts.

4.3.2. Modifying the test policy file from the outline view

In addition to navigation, the outline view also helps you modify your test policy file. Right-click on any of the elements to bring up a menu of options relative to the specific item.



The options available from this menu, depending on the type of element, are:

Add

This option allows you to add a new filter under the *rules* section. This will open a sub-menu of filter types to choose from. Click one of the choices to have it added to your policy file in the appropriate location. You can then modify the inserted filter to meet your needs.

Delete

This option will delete the element and any child elements below it.

Documentation

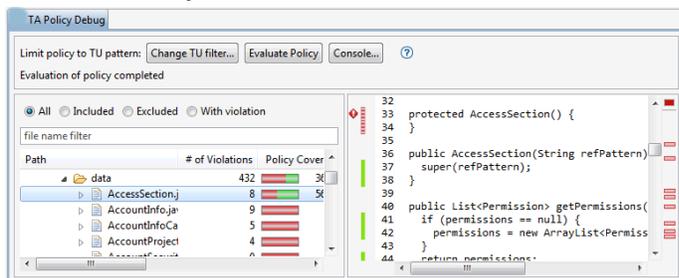
This option will open the product documentation for the selected node in your web browser.

Evaluate

This option will evaluate the selected rule or named filter against your current project.

4.4. Debugging your test policy file

The *Policy Debug* view evaluates your entire test policy file against your current project, then displays the results in the project's file tree. To run the evaluation, choose your project's language and click the **Evaluate Policy** button.

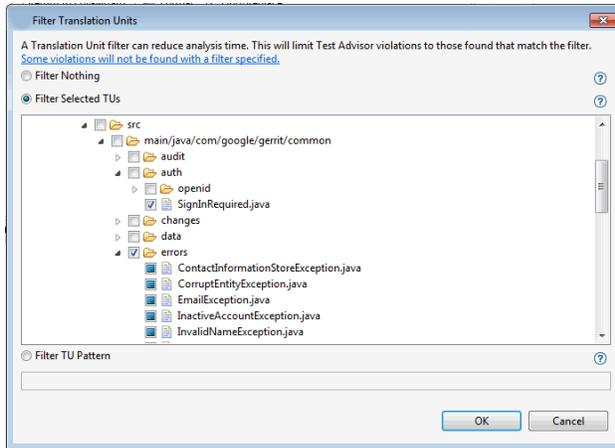


The file tree lists all of the files and directories in your project, and displays the number of violations and policy coverage percentage in each. You can filter the displayed directories using the *Included*, *Excluded*, or *With violation* radio buttons. Select an individual file to display the text in the view pane.

4.4.1. Translation Unit Filters

If your project has a large number of translation units (TUs), the analysis can take a long time to complete. To compensate for this, you can apply a TU filter to limit the number of translation units being analyzed by the process and reduce the time to start the debugger.

If your project contains over 100 translation units, you will be prompted to add a TU filter as soon as you open the policy editor. Otherwise, to open the *Filter Translation Units* dialog, click on the **Change TU filter** button. See Figure 4.1, "Filter Translation Units" for an example of this view.

Figure 4.1. Filter Translation Units

You can use the file tree to select the TUs you would like to include in the analysis, or enter a Filter TU Pattern, as defined in the *Test Advisor 2020.12 User and Administrator Guide*.

**Note**

Because the TU filter limits the amount of files that will be analyzed, some defects may not be found. Therefore, expanding the set of included translation units will generally yield more accurate results.

If you require a complete set of defects from your analysis, do not include a TU filter. This will require more time to complete the analysis (perhaps overnight). Once completed, further evaluations will complete much quicker as long as you don't exit from the editor/debugger.

For additional information on effectively implementing Translation Unit filters, see Section 4.5, “Translation Unit Filter reference”.

4.5. Translation Unit Filter reference

When running an analysis with a Translation Unit Filter, Test Advisor results may be different than the results of analysis without a Translation Unit Filter. Results of filters in the Test Advisor policy may differ due to information which is omitted based on the specific Translation Unit Filter used. In particular, the following filters are affected:

- `UnconditionalTerminateCallASTNodeFilter`
- `ImpactedDateLineFilter`
- `AffectedDateLineFilter`

For these filters, calls to functions defined in omitted Translation Units are ignored during analysis.

Because the results of the policy analysis may change when a Translation Unit Filter is used, a complete run of analysis without a Translation Unit Filter is necessary to guarantee a correct set of defects from

Test Advisor. However, simply expanding the set of included Translation Units may yield more accurate results in certain cases when the above-mentioned filters are used.

Chapter 5. Using the Guided Test Advisor Policy Creation Wizard

Table of Contents

5.1. Introduction	33
5.2. Code Age Thresholds	34
5.3. Violation Criteria	34
5.4. Filters	35

This section illustrates the typical end-to-end workflow for creating a new Test Advisor Policy file using the wizard. You can access the Policy Creation Wizard from the Analysis Settings screen, by selecting **Create New Policy Using Wizard** from the drop-down menu.

The wizard consists of a single dialog, with collapsible sections for introductory information, code age thresholds, violation criteria, and filters. Upon completing the configuration of all of the sections, click **Open Policy in Test Advisor Policy Editor**. From there you can further edit your policy file, or click **Return to cov-wizard** to use your policy file with Coverity Wizard for Test Advisor - Development Edition.

5.1. Introduction

Be sure to read and understand the information in this section before configuring the options in the subsequent sections.

A Test Advisor policy file expresses the testing related rules you wish to enforce in your project. The policy file defines the conditions under which violations will be reported for insufficient test coverage in your code.

This wizard will guide you through the process of creating and customizing a policy file for a range of standard scenarios. The policy file code will be automatically generated for you.

In the interest of simplicity, this wizard will limit you to creating one definition of a violation type (that is, a single rule) per policy file. Test Advisor policy files can contain any number of rules, however, and you can copy rules generated via this wizard into other policy files should you wish to do so.

5.2. Code Age Thresholds

Figure 5.1. Configuring code age thresholds

Code Age

Code that has been modified more recently is less likely to have been extensively tested, and is thus more likely in general to contain bugs.

In your policy file, you are required to specify threshold dates, which will be used to display code age indicators in Coverity Connect, and in the Test Advisor policy editor/debugger.

- Code modified before the Old Date Cutoff threshold will be marked with a single bar in the margin.
- Code modified between the Old Date Cutoff and the Recent Date Cutoff threshold will be marked with two bars.
- Code modified after the Recent Date Cutoff will be marked with three bars.

Typically, you will use the date of your most recent release as the Recent Date Cutoff, and the date of the release previous to that as the Old Date Cutoff.

Code age indicator

Old date cutoff

1/ 1/2013

Recent data cutoff

1/ 1/2014

The diagram illustrates the code age indicator logic. It shows two vertical lines representing the 'Old date cutoff' (1/ 1/2013) and the 'Recent data cutoff' (1/ 1/2014). The area to the left of the Old date cutoff is labeled 'Old code' and contains a single vertical bar. The area between the Old date cutoff and the Recent data cutoff is labeled 'Medium-aged code' and contains two vertical bars. The area to the right of the Recent data cutoff is labeled 'New code' and contains three vertical bars. Arrows indicate the direction of the cutoffs: the Old date cutoff points left, and the Recent data cutoff points right.

The *Code Age Thresholds* section helps you set cutoff dates to determine whether the code being analyzed is "old," "medium-aged," or "new." This helps you identify which test violations are most important, as newer code is more likely to contain bugs, and thus should not go untested.

Use the *Old date cutoff* and *Recent date cutoff* fields to specify the threshold dates you wish to use for each. Typically, the date of your most recent release will serve as the Recent Date Cutoff, and the date of the previous release will be the Old Date Cutoff.

5.3. Violation Criteria

Figure 5.2. Configuring violation criteria

Violation Criteria

A rule defines a type of testing violation that will be detected and reported. In this section, you will choose a name for the type of violation you wish to report, and specify the basic conditions under which the violation will be generated. The conditions can be further refined by adding one or more filters.

Violation name:

Violation name: ?

This name will be shown as the checker name for all instances of this violation, so it should be suitably descriptive to aid the developer in understanding the nature of the violation. A name such as FUNCTION_INSUFFICIENTLY_TESTED or FILE_INSUFFICIENTLY_TESTED is usually sufficient, unless you have many rules, in which case the names should clearly distinguish between the different types of violations.

Violation criteria:

Report this violation for: Functions ?

Files

Report this violation when the percentage of covered lines in a function is less than: % ?

The *Violation Criteria* section helps you specify what type of testing violation will be captured by the Test Advisor policy. Complete the following configuration steps:

1. Create a name for the violation in the *Violation name* field.

This name will be shown as the checker name for all instances of the violation. Be sure that it is descriptive enough to aid users in understanding the nature of the violation (for example, `FUNCTION_INSUFFICIENTLY_TESTED`).

2. Select if you want to see test violations on a per-function or per-file basis. If you select *Functions*, the filter criteria in this rule will test each function to see if it violates the policy. When *Files* is selected, the filter criteria will test each file to see if it violates the policy.

Any function or file that violates the rule will then result in a test violation.

3. Specify what percentage of lines must be covered under the rule. This percentage applies to the lines of code that remain after any specified filters have been applied.

If test coverage is less than this percentage of any relevant file or function, a violation will be reported.

5.4. Filters

Figure 5.3. Configuring filters

The screenshot shows the 'Filters' configuration section. It starts with a dropdown arrow and the label 'Filters'. Below this is a paragraph of explanatory text: 'You can choose to apply one of more of the following filters in order to cause the violation to be reported only under more specific circumstances. The following list contains the most commonly used filters. For more information on advanced filters that can be added by editing the policy file in the Test Advisor policy editor/debugger, please refer to the [documentation](#).' This is followed by a checkbox labeled 'Include/exclude files or subdirectories' with a help icon. Underneath are two text input areas. The first is labeled 'Regular expression of files or directories to include:' and contains a list with entries '/src' and '/utils'. To the right of this list are buttons for 'Add', 'Edit', 'Remove', 'Up', and 'Down'. The second input area is labeled 'Regular expression of files or directories to exclude:' and contains the entry '/src/test', with similar 'Add', 'Edit', 'Remove', 'Up', and 'Down' buttons to its right. At the bottom of the section are three more checkboxes, each with a help icon: 'Report violations only for functions modified after the recent date cutoff', 'Report violations only for functions impacted after the recent date cutoff', and 'Report violations only for functions executed by the test suite'.

The conditions of your violation criteria can be further refined by adding one or more filters. Several of the most commonly used filters are configurable in the *Filters* section.

Using the Guided Test Advisor Policy Creation Wizard

You can include any of the listed filters by checking the box next to its name. Each filter has a help button (🔗) which explains its impact.

Chapter 6. Troubleshooting Coverity Wizard

This section provides workarounds and procedures for solving common problems that might occur when you run Coverity Wizard.

Using the Tab key on Mac OSX systems

On Windows and Linux systems, pressing the Tab key will allow you to cycle through all of the controls on the page. However, in Mac OS X, the default operating system setting is to allow the focus to only land on text fields and lists, bypassing buttons and other controls. To change this setting to allow keyboard navigation and activation of buttons and combo boxes:

1. Open the Apple menu and go to *System Preferences -> Keyboard*
2. Go to the *Keyboard Shortcuts* tab.
3. Under *Full Keyboard Access*, change the setting from *Text boxes and lists only* to *All controls*.

Troubleshooting the Visual Studio IDE

If you are having trouble with the IDE shortcut feature using Visual Studio please refer to the following:

1. If Visual Studio requires elevated permission (i.e. *Run as Administrator* is selected under *Privilege Level* in the *Compatibility* tab of the Visual Studio shortcut properties window), then you must run Coverity Wizard as Administrator level privileges.
2. Enabling *Instrument* mode in the *Advanced* tab is discouraged when using Visual Studio. When trying *Instrument* mode, some path settings may be required, as described in the *Coverity Desktop 2020.12 for Microsoft Visual Studio: User Guide*.

Appendix A. Coverity Glossary

Table of Contents

Glossary	38
----------------	----

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.
Acyclic Path Count	<p>The number of execution paths in a function, with loops counted one time at most. The following assumptions are also made:</p> <ul style="list-style-type: none">• <code>continue</code> breaks out of a loop.• <code>while</code> and <code>for</code> loops are executed exactly 0 or 1 time.• <code>do...while</code> loops are executed exactly once.• <code>goto</code> statements which go to an earlier source location are treated as an exit. <p><i>Acyclic (Statement-only) Path Count</i> adds the following assumptions:</p> <ul style="list-style-type: none">• Paths within expressions are not counted.• Multiple case labels at the same statement are counted as a single case.
advanced triage	<p>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 <i>Bug</i> classification or a <i>Fix Required</i> action) for that CID, regardless of whether the streams belong to the same project.</p> <p>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:</p>

- 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.

analysis annotation

A marker in the source code. An analysis annotation is not executable, but modifies the behavior of Coverity Analysis in some way.

Analysis annotations can suppress false positives, indicate sensitive data, and enhance function models.

Each language has its own analysis annotation syntax and set of capabilities. These are not the same as the syntax or capabilities available to the other languages that support annotations.

- For C/C++, an analysis annotation is a comment with special formatting. See code-line annotation and function annotation.
- For C# and Visual Basic, an analysis annotation uses the native C# attribute syntax.
- For Java, an analysis annotation uses the native Java annotation syntax.

Other languages do not support annotations.

annotation

See analysis 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. For details about checkers, see *Coverity 2020.12 Checker Reference*.

Coverity Glossary

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 identifier (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-line annotation	<p>For C/C++, an analysis annotation that applies to a particular line of code. When it encounters a code-line annotation, the analysis engine skips the defect report that the following line of code would otherwise trigger.</p> <p>By default, an ignored defect is classified as <i>Intentional</i>. See "Models and Annotations in C/C++" in the <i>Coverity Checker Reference</i>.</p> <p>See also function annotation.</p>
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 <i>instances</i> (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.

Coverity Connect

A Web application that allows developers and managers to identify, manage, and fix issues found by Coverity analysis and third-party tools.

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 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 Dynamic Analysis reports during a single invocation of the 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 <code>else</code> branch of an <code>if</code> statement which tests that condition cannot be reached on that path. Any defects found in the <code>else</code> 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. In C or C++ source, you might also use code-line annotations 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	For C/C++, an analysis annotation that applies to the definition of a particular function. The annotation either suppresses or enhances the effect of that function's model. See "Models and Annotations in C/C++" in the <i>Coverity Checker Reference</i> . See also code-line 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 <code>--dir</code> 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.

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	<p>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, Security, and Test Advisor dashboards in Coverity Connect provide high-level metrics on each type of issue.</p> <p>Note that this glossary includes additional entries for the various types of issues, for example, an inspected issue, issue category, and so on.</p>
issue category	<p>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.</p> <p>Examples:</p> <ul style="list-style-type: none">• <code>Memory - corruptions</code>• <code>Incorrect expression</code>• <code>Integer overflow Insecure data handling</code> <p>Impact tables in the <i>Coverity 2020.12 Checker Reference</i> list issues found by checkers according to their category and other associated checker properties.</p>

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# and Visual Basic, 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	<p>In Coverity Analysis of the code for a compiled language—such as C, C++, C#, Java, or Visual Basic—a model represents a function in the application source. Models are used for interprocedural analysis.</p> <p>Each model is created as each function is analyzed. The model is an abstraction of the function’s behavior at execution time; for example, a model can show which arguments the function dereferences, and whether the function returns a null value.</p> <p>It is possible to write custom models for a code base. Custom models can help improve Coverity's ability to detect certain kinds of bugs. Custom models can also help reduce the incidence of false positives.</p>
modeling primitive	<p>A modeling primitive is used when writing custom models. Each modeling primitive is a function stub: It does not specify any executable code, but when it is used in a custom model it instructs Coverity Analysis how to analyze (or refrain from analyzing) the function being modeled.</p> <p>For example, the C/C++ checker CHECKED_RETURN is associated with the modeling primitive <code>_coverity_always_check_return_()</code>. This primitive tells CHECKED_RETURN to verify that the function being analyzed really does return a value.</p> <p>Some modeling primitives are generic, but most are specific to a particular checker or group of checkers. The set of available modeling primitives varies from language to language.</p>

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.
outstanding defects count	The sum of security and non-security defects count.
outstanding non-security defects count	The sum of non-security defects count.
outstanding security defects count.	The sum of security defects count.

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.

primitive In the Java language, elemental data types such as strings and integers are known as *primitive types*. (In the C-language family, such types are typically known as *basic types*).

For the function stubs that can be used when constructing custom models, see modeling primitive.

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 Coverity 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	<p>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.</p> <p>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.</p>
snapshot scope	<p>Determines the snapshots from which the CID are listed using the <i>Show</i> and the optional <i>Compared To</i> fields. The show and compare scope is only configurable in the <i>Settings</i> menu in <i>Issues:By Snapshot</i> views and the snapshot information pane in the <i>Snapshots</i> view.</p>
source	<p>An entry point of untrusted data. Examples include environment variables, command line arguments, incoming network data, and source code.</p>
static analysis	<p>Analysis of software code without executing the compiled program. See also dynamic analysis.</p>
status	<p>Describes the state of an issue. Takes one of the following values: <i>New</i>, <i>Triaged</i>, <i>Dismissed</i>, <i>Absent</i> <i>Dismissed</i>, or <i>Fixed</i>.</p>
store	<p>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.</p>
stream	<p>A sequential collection of snapshots. Streams can thereby provide information about software issues over time and at a particular points in development process.</p>

T

tainted data	<p>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.</p>
translation unit	<p>A translation unit is the smallest unit of code that can be compiled separately. What this unit is, depends primarily on the language: For example, a Java translation unit is a single source file, while a C or C++ translation unit is a source file plus all the other files (such as headers) that the source file includes.</p> <p>When Coverity tools capture code to analyze, the resulting intermediate directory contains a collection of translation units. This collection includes source files along with other files and information that form the</p>

context of the compilation. For example, in Java this context includes bytecode files in the class path; in C or C++ this context includes both preprocessor definitions and platform information about the compiler.

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.

triage store A repository for the current and historical triage values of CIDs. In Coverity Connect, each stream must be associated with a single triage store so that users can triage issues (instances of CIDs) found in the streams. Advanced triage allows you to select one or more triage stores to update when triaging a CID in a Coverity Connect project.

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 2020.12 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. For details, see the *Coverity 2020.12 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. Coverity Legal Notice

Table of Contents

B.1. Legal Notice	50
-------------------------	----

B.1. 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 © 2020 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.

Copyright © Microsoft Corporation. All rights reserved.

Oracle and Java are registered trademarks of Oracle and/or affiliates. Other names may be trademarks of their respective owners.

"MISRA", "MISRA C" and the MISRA triangle logo are registered trademarks of MISRA Ltd, held on behalf of the MISRA Consortium. © MIRA Ltd, 1998 - 2013. All rights reserved. The name FindBugs and the FindBugs logo are trademarked by The University of Maryland.

Other names and brands may be claimed as the property of others.

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 `software-integrity-support@synopsys.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 `software-integrity-support@synopsys.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 `software-integrity-support@synopsys.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 the JaCoCo package instead.

For information about using JaCoCo, see the description for `cov-build --java-coverage` in the *Command Reference*.

LLVM/Clang subproject

Copyright © All rights reserved. Developed by: LLVM Team, University of Illinois at Urbana-Champaign (<http://llvm.org/>). 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.

Rackspac Threading Library (2.0)

Copyright © Rackspac, 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 © 2020 Synopsys Inc. All rights reserved worldwide. (www.synopsys.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>.

Apache Software License, Version 1.1

Copyright © 1999-2003 The Apache Software Foundation. All rights reserved.

Redistribution and use in source and binary forms, with or without modification, are permitted provided that the following conditions are met:

Coverity Legal Notice

1. Redistributions of source code must retain the above copyright notice, this list of conditions and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions and the following disclaimer in the documentation and/or other materials provided with the distribution.
3. The end-user documentation included with the redistribution, if any, must include the following acknowledgement: "This product includes software developed by the Apache Software Foundation (<http://www.apache.org/>)."

Alternately, this acknowledgement may appear in the software itself, if and wherever such third-party acknowledgements normally appear.

4. The names "The Jakarta Project", "Commons", and "Apache Software Foundation" must not be used to endorse or promote products derived from this software without prior written permission. For written permission, please contact apache@apache.org.
5. Products derived from this software may not be called "Apache" nor may "Apache" appear in their names without prior written permission of the Apache Group.

THIS SOFTWARE IS PROVIDED ``AS IS" AND ANY EXPRESSED OR IMPLIED WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL THE APACHE SOFTWARE FOUNDATION OR ITS CONTRIBUTORS BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL, EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY, WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

Apache License Version 2.0, January 2004 <http://www.apache.org/licenses/>
Licensed under the Apache License, Version 2.0 (the "License"); you may not use this file 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.

Results of analysis from Coverity and Test Advisor represent the results of analysis as of the date and time that the analysis was conducted. The results represent an assessment of the errors, weaknesses and vulnerabilities that can be detected by the analysis, and do not state or infer that no other errors, weaknesses or vulnerabilities exist in the software analyzed. Synopsys does NOT guarantee that all errors, weakness or vulnerabilities will be discovered or detected or that such errors, weaknesses or vulnerabilities are discoverable or detectable.

SYNOPSYS AND ITS SUPPLIERS DISCLAIM ALL WARRANTIES, CONDITIONS AND REPRESENTATIONS, EXPRESS, IMPLIED OR STATUTORY, INCLUDING THOSE RELATED

TO MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, ACCURACY OR COMPLETENESS OF RESULTS, CONFORMANCE WITH DESCRIPTION, AND NON-INFRINGEMENT. SYNOPSIS AND ITS SUPPLIERS SPECIFICALLY DISCLAIM ALL IMPLIED WARRANTIES, CONDITIONS AND REPRESENTATIONS ARISING OUT OF COURSE OF DEALING, USAGE OR TRADE.