Suppressing the Reporting of Acceptable Violations

This topic explains how to prevent C/C++test from reporting selected static analysis violations.

Sections include:

About Suppressions

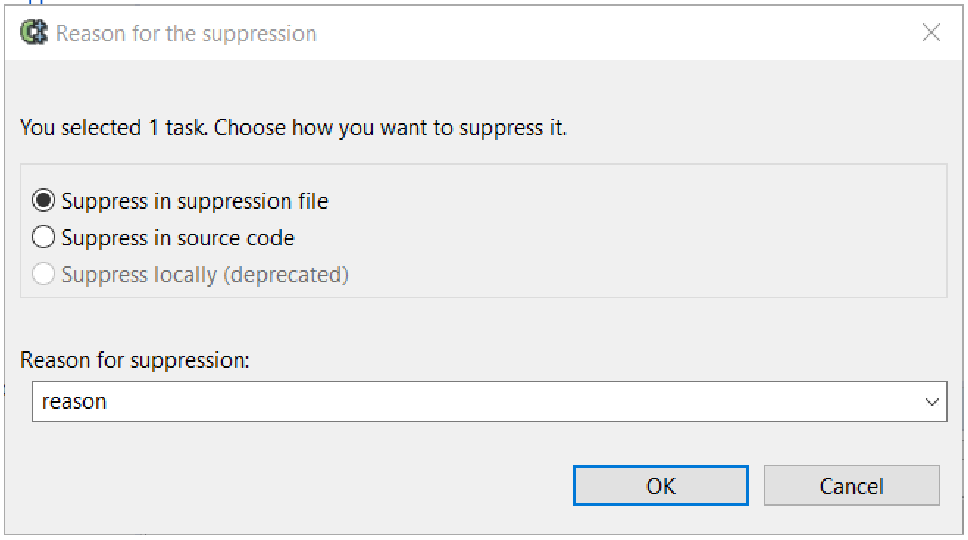
Suppressions are used to prevent C/C++test from reporting selected occurrences of a static analysis violations. You use suppressions for situations when you generally want to follow a rule, but decide to ignore specific occurrences of the reported task.

Suppressions can be stored in source code or in local suppression files, which can be checked in the source control system so hat they are shared across the team. You can use the Parasoft GUI to create suppressions or manually add information about suppressions in your source code or suppression files.

If you are using the MISRA Compliance Pack for DTP, see the DTP User Guide to learn more about how suppressions are handled by MISRA compliance reporting on DTP.

Defining Suppressions in the GUI

To suppress a static analysis task that is shown in the Quality Tasks view:

1. Right-click a Quality Tasks view item that represents the task you want to suppress, then choose **Suppress Task...** from the menu.
   To suppress more than one task, right-click the node that represents a group of tasks (a rule category, a specific rule, a file), then choose **Suppre ss All Tasks**
2. Choose where the suppressions will be stored. You can select one of the following options:
   - **Suppress in suppression file** - The suppression will be stored in *a parasoft.suppress* file located in the same directory as the corresponding source file. See Defining Suppressions in Suppression Files for details.
   - **Suppress in source code** - The selected task will be suppressed in code and shared across the team when checked in your source control system.
   - **Suppress locally (deprecated)** - The suppression will be stored locally in the deprecated XML-style format. See Using the Deprecated Suppression Format for details.
   
   Reason for the suppression

   You selected 1 task. Choose how you want to suppress it.

   [✓ Suppress in suppression file]
   Suppress in source code
   Suppress locally (deprecated)

   Reason for suppression: reason

3. Enter the reason for suppression.
4. Click **OK** to complete suppression. The suppressed task(s) will be removed from the Quality Tasks view.

Defining Suppressions in Suppression Files

If you choose to store a suppression in a file, it will be stored in a *parasoft.suppress* file located in the same directory as the source file that contains the rule violation. You can define in-file suppressions in one of the following ways:

- By manually creating *parasoft.suppress* files where you can add suppression entries. If you manually create a suppression file, ensure it is located in the same directory as the source file that contains the task you want to suppress.
- By choosing the **Suppress in suppression file** option in the GUI (see Defining Suppressions in the GUI).Newly created suppression files are automatically included in your project and displayed in the file tree in the IDE along with other project files.

We recommend that suppression files be checked in your source control system. This allows you to share information about suppressions with other team members and easily review the suppressions on a branch in your SCM repository before merging the code into the main stream of development, such as "master", "trunk', etc.

Use the following format to add suppression entries to *parasoft.suppress* files:

```
suppression-begin
file: Account.cpp                  (required)
line: 12                           (optional)
rule-id: CODSTA-123                (optional)
message: Exact violation message   (optional)
reason: Approved                   (optional)
author: devel                      (optional)
date: 2020-09-21                   (optional)
suppression-end
```

Example

At a minimum, you must specify the source file where the problem was detected. This will suppress all findings reported for the specified file. In the following example, all findings detected in the A*ccount* file will be suppressed:

```
suppression-begin
file: Account.cpp
suppression-end
```

Other attributes are optional and help you fine-tune the suppression. In the following example, all findings that the PB.TYPO.TLS rule detected in the *Accou nt* file are suppressed, regardless on which code line they occur:

```
suppression-begin
file: Account.cpp
rule-id: PB.TYPO.TLS
suppression-end
```

The `line` attribute should be used with caution as it may invalidate the suppression if the code is moved to another line when the source file is modified.

Defining Suppressions in Source Code

When suppressions are defined in source code:

- You ensure that the same suppressions are applied whenever you or a team member tests that code.
- You can add code comments explaining each suppressions, so the reason for each suppression is always clear when you or team members are reviewing the code.
- You gain fine-grained control over which rules are enforced at the file, class, or line level.

You can define suppressions in source code either by choosing the appropriate option in the GUI (see Defining Suppressions in the GUI) or by manually them manually in source code – using the following suppression syntax.

Line Suppression

```
<suppression keyword> [<rule category> | <rule category> . <rule id> | <rule category > - <rule severity> | ALL
] <suppression comment>
```

Line Suppression Examples

```
//parasoft-suppress CODSTA "suppress all rules in category CODSTA"

// parasoft-suppress CODSTA.NEA "suppress rule CODSTA.NEA"

// parasoft-suppress CODSTA-1 "suppress all rules in category CODSTA with severity level 1"

// parasoft-suppress ALL "suppress all rules"

// parasoft-suppress CODSTA FORMAT.MCH JAVADOC-3 "suppress all rules in category CODSTA and rule FORMAT.MCH and
all rules in category JAVADOC with severity level 3"
```

Block Suppression

```
<begin suppression keyword> [<rule category> | <rule category> . <rule id> | <rule category > - <rule severity>
| ALL ] <suppression comment>

 ..... source code block .....

<end suppression keyword> [<rule category> | <rule category> . <rule id> | <rule category > - <rule severity> |
ALL ] <suppression comment>
```

Block Suppression Examples

```
// parasoft-begin-suppress CODSTA "begin suppress all rules in category CODSTA"
.....
// parasoft-end-suppress CODSTA "end suppress all rules in category CODSTA"

// parasoft-begin-suppress CODSTA.NEA "begin suppress rule CODSTA.NEA"
.....
// parasoft-end-suppress CODSTA.NEA "end suppress rule CODSTA.NEA"

// parasoft-begin-suppress CODSTA-1 "begin suppress all rules in category CODSTA with severity level 1"
......
// parasoft-end-suppress CODSTA-1 "end suppress all rules in category CODSTA with severity level 1"

//parasoft-begin-suppress ALL "begin suppress all rules"
.....
// parasoft-end-suppress ALL "end suppress all rules"

// parasoft-begin-suppress CODSTA FORMAT.MCH "begin suppress all rules in category CODSTA and rule FORMAT.MCH"
.....
// parasoft-end-suppress CODSTA FORMAT.MCH "end suppress all rules in category CODSTA and rule FORMAT.MCH"

// parasoft-begin-suppress CODSTA "begin suppress all rules in category CODSTA"
.....
// parasoft-end-suppress CODSTA-1 "end suppress all rules in category CODSTA with severity level 1; however
rules with severity level 2-5 in category CODSTA are still suppressed."
.....
// parasoft-end-suppress CODSTA "end suppress all rules in category CODSTA"

// parasoft-begin-suppress ALL "begin suppress all rules"
.....
// parasoft-end-suppress CODSTA FORMAT-1 "end suppress all rules in category CODSTA and all rules in category
FORMAT with severity level 1; however, others rules in CODSTA and FORMAT-1 are still suppressed."
.....
// parasoft-end-suppress ALL "end suppress all rules"

//parasoft-begin-suppress ALL "begin suppress all rules, since no end suppression comment, all rules will be
suppressed starting from this line"
```

The following suppresion directives are deprecated:

- `parasoft off`: Suppresses the reporting of all static analysis violations that occur between this line of code and the end of the file.
- `parasoft on`: Unsuppresses the reporting of all static analysis violations that occur between this line of code and the end of the file.
- `parasoft suppress/unsuppress [line <linewildcard>][class <classnamewildcard>][file <filenamewildcard>] [item <ruleid>][type <severity>] [reason <comment>]`: Creates a suppression with the desired parameters.

Defining Line Suppressions Based on Regex Patterns

You can configure C/C++test to automatically suppress static rule violations that are detected on lines that match a regular expression pattern. This may be useful when you want to suppress tasks that are difficult to suppress using the in-line or in-code suppressions, such as Qt macros.

To define regular expressions patterns to specify the code lines:

1. Create an advanced settings file that includes the following C/C++test's advanced settings:

```
cpptest.result.line.suppressions.enabled=true          // Enables creating regex-based suppressions.
cpptest.result.line.suppressions.pattern=[regex;regex] // Specifies a semicolon-separated list of regex
patterns.
```

2. Specify the path to the advanced settings. See Configuring an Advanced Settings File.

Static rule violations that occur on lines that match the configured regex pattern(s) will be suppressed. For example, the following configuration will suppresses all findings detected on code lines that contain "Q_" anywhere on the line.

```
cpptest.result.line.suppressions.enabled=true
cpptest.result.line.suppressions.pattern=.*Q_.*
```

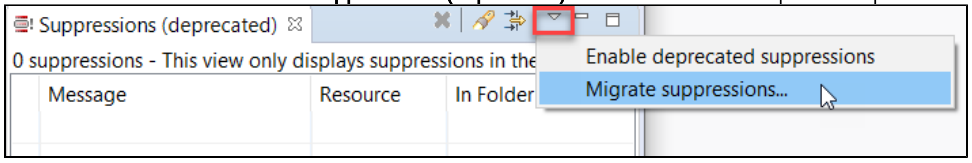
Handling Deprecated Suppressions

Storing suppressions in the XML-style format locally or on Team Server is deprecated. Starting with C/C++test 2020.2, suppressions are created either directly in code or in the new in-file format (see Defining Suppressions in Suppression Files). If there are suppressions in the deprecated format available for your project (typically, when you upgrade C/C++test to version 2020.2 or later and you have created suppressions with previous versions of C/C++test) they are still applied to your code and you can review them in the deprecated Suppressions view.

Migrating Deprecated Suppressions

You can migrate local suppressions to a new format using C/C++test's capabilities. If you need to migrate suppressions stored on Team Server, please contact Parasoft Support.

To migrate deprecated local suppressions:

1. Choose **Parasoft > Show View > Suppressions (deprecated)** from the IDE menu to open the deprecated Suppressions view.

Suppressions (deprecated)
0 suppressions - This view only displays suppressions in the
Message
Resource
In Folder
Enable deprecated suppressions
Migrate suppressions...
2. Choose **Migrate suppressions...** from the menu.
3. Follow the instructions displayed in the dialog that opens.

Migration has the following consequences:

- Suppressions in the deprecated format are converted into plain text and saved in *parasoft.suppress* files located in the same location as the corresponding source file.
- A backup file that contains suppressions in the deprecated format is created. The location of the file is printed to the console.
- Suppressions in the deprecated format are removed from the workspace and from the deprecated Suppressions view.

If you should want to restore deprecated suppressions from the backup file:

1. Close your IDE.
2. Go to the location where the backup file is stored (the location is printed to the console). It contains a *local_suppressions_backup.zip* file.
3. Extract the contents of *local_suppressions_backup.zip* to the same location.

You can re-migrate restored suppressions on next IDE startup.

Enabling Deprecated Suppressions

Enabling deprecated suppressions allows you to create suppressions in the deprecated format and store them locally or on Team Server.

1. Choose **Parasoft > Show View > Suppressions (deprecated)** from the IDE menu to open the deprecated Suppressions view.
2. Enable the **Enable deprecated suppressions** option in the menu.