Configuring and Running Post-Commit Code Review Scans

This topic explains how to set up and run a post-commit code review scan that scans the source control system, identifies new/modified code that has been checked in, and matches the code with designated reviewers.

Sections include:

Related Topics

- For details on **pre-commit code review**, which is for teams who want to review code *before* it is committed to source control, see Configuring and Running Pre-Commit Code Review Scans.
- For details on pre-commit vs. post-commit vs. task-driven workflows, see Workflow Overview.
- For details on general code review configuration options (options that apply to both pre-commit and post-commit processes, see General Code Review Configuration.

Configuration Overview

Configuring a post-commit code review requires:

- On the Parasoft Test server installation:
  - Configuring a Code Review Test Configuration as described below.
  - Configuring the command-line execution of this Test Configuration as described below.
  - Scheduling automated scans as described below.
  - Ensuring that the projects containing the reviewed code are available within Parasoft Test. These projects should be checked out from source control.
- On all team Parasoft Test installations —including desktop and server installations:
  - Setting up the appropriate preferences on all team Parasoft Test installations as described in General Code Review Configuration.

Configuring a Test Configuration for Post-Commit Code Reviews

For post-commit processes, a code review Test Configuration runs on the server each night to scan the source control system, identify new/modified code that has been checked in, and match the code with designated reviewers.

Important

The default Code Review Test Configuration settings must be reviewed and customized before you run Code Review.

To configure a Test Configuration that detects code changes and prepares them for review:

1. Open the Test Configurations dialog by choosing **Parasoft> Test Configurations** or by choosing **Test Configurations** in the drop-down menu on the **Test Using** toolbar button.
2. Duplicate an existing Test Configuration (such as **Built-in> Code Review> Post-Commit**) or create a new one.
3. In the **Scope** tab, choose **Test files added or modified in the last ___ days**. This determines what files are prepared for code review. For example, if the Code Review scanner will run daily, enter 1 in the **days** field. If it will run weekly, enter 7.
   - Alternatively, you can configure it to prepare all files modified within a specified time period. Instead of choosing **Test files added or modified in the last ___ days**, choose the **Test only files added or modified since the cutoff date** option and the **and added or modified before** option, then specify the desired time range.
4. Also in the **Scope** tab, review the **File Filters> Path options** settings and adjust them if desired. For details on these Test Configuration settings, see Configuring Test Configurations).
5. In the **Common** tab, check **Update projects from source control**.
   - The project should be fully updated from the source control repository before the code review scan is performed. If you have an external system that updates resources before the code review scan, you do not need to enable the **Update projects from source control** option. If you do not have another system performing updates, we strongly recommend that you enable this option.
6. If your reporting preferences are not set to use a unique session tag for code review scans, go to the **Common** tab, enable **Override Session Tag**, then choose one of the preconfigured identifiers, or specify your own. This is the tag that will be assigned to all code reviews that stem from this Test Configuration.
   - *To ensure proper code review reporting, you must configure a session tag for your code review scans and use that session tag in DTP to specify which code reviews are associated with particular DTP projects.*
7. At the top of the **Code Review** tab:
   a. Check **Enable Code Review Scanner**.
   b. Check **Generate comprehensive report (includes all scanners)** if you want the report to include code review results from all available team scanners. If this is not enabled, the report will include only results for the session tag set for the current Test Configuration.

    c. Enable **Auto publish reviews** if you want review tasks to be "published" (uploaded) automatically after this Test Configuration is run. If you use the -publishteamserver option with a nightly run, tasks will be "published" regardless of this setting.
    d. From the **Generate tasks with priority** box, indicate the priority that should be assigned to all code review tasks that are created using this Test Configuration.
8. In the **Authors**, **Reviewers**, **Monitors**, and **Filters** tabs, define how you want your code reviews assigned. Reviewers and monitors can be assigned to specific authors, or to specific project areas.
    - In the **Authors** tab, define the list of developers who are writing code that you want reviewed. For each author, specify an author name and a source control login (if the author's source control login is different than the author's name).
        - Your list of authors can include all of your developers, or only your junior developers.
        - If the developer who commits a code change is not defined in this tab, the change will be marked as coming from an 'undefined author'.
        - You do not need to map authors to reviewers or monitors here. These fields are provided for backwards compatibility with earlier releases. If you don't want to define every developer in the **Authors** tab, you can 1) enable the **Filter** tab's **Accept all (also undefined) authors for reviewed paths**, and then 2) Use the **Reviewers** tab to define which reviewers should review different parts of the code.
    - In the **Reviewers and Monitors** tabs, specify which authors and/or project areas you want each reviewer or monitor to cover.
        - Reviewers examine, comment on, and approve code changes. Monitors supervise the entire process to ensure that revisions are being reviewed and then corrected in a timely manner. They do not need to perform any reviews, but can add comments to the revisions or reviews. This role is optional.
        - Paths are defined in logical (workspace) path convention. Wildcards are allowed. See the "Filter Tips and Examples" box below for more details and examples.
        - You can define reviewers and monitors without mapping them to any particular path or author. Such users will be not assigned to any package automatically, but they will be included in the report and authors will be able to select them in the Code Review Assistant dialog.
        - These tabs use OR logic (not AND logic). In other words, you define the name of a reviewer (or monitor), then the authors and review paths you want that person to review (or monitor). Then, if new code comes from either the specified authors OR the specified paths, it will be assigned to the named reviewer or monitor.
9. (Optional) In the **Code Review> Filters** tab, modify the following options if desired:
    - **Ignore changes within suppressed blocks:** Enable this option if you want the code review scan to ignore all differences that occur between "codereview-begin-suppress" and "codereview-end-suppress" markers. These are the same markers that are used to prevent the compare editor from displaying differences within specific blocks of code (see Hiding Differences for Specific Pieces of Code for details).
    - **Differences**: If you want the code review scan to ignore all current source vs. previous source differences that match a certain pattern, specify the appropriate regular expression here. For example, you might use this to prevent any differences in automatically-generated comments from being flagged as requiring code review.
    - **Post to Pre-Commit matching:** Contains options for hybrid pre-commit and post-commit code review processes, where developers are expected to commit code for review prior to checking but post-commit scans are also used to validate that this process is being followed.
        - **Post to Pre-Commit matching:** If you want the post-commit scan to ignore all pre-commit scan vs. post-commit scan differences that match a certain pattern, then specify the appropriate regular expression here. For example, you might use this to prevent differences in automatically-generated CVS headers added upon checkin from being flagged for post-commit code review. See Running Post-Commit Scans with a Pre-Commit for more details.
        - **Pre-commit search range in days:** If you want to customize the timeframe used to determine which pre-commit tasks correspond to post-review tasks, change this setting. See Running Post-Commit Scans with a Pre-Commit for more details.
    - **Accept all (also undefined) authors for reviewed path:** If you don't want to define every developer, you can 1) enable the **Accept all (also undefined) authors for reviewed paths**, and then 2) Use the **Reviewers** tab to define which reviewers should review different parts of the code.
10. Click **Apply** to commit the new Test Configuration.
11. Test the Test Configuration by selecting one of your projects in the project tree, then running the Test Configuration.

Preparing Code for Review - Automated Scanner Execution

For post-commit code reviews, Parasoft test is typically scheduled to run in command-line mode at a regularly-scheduled interval (for example, daily). These tests execute the team's Code Review Test Configuration. Each time a test runs, the Code Review scanner scans the designated source control repositories for new/modified code, then prepares any detected changes for review.

cli setup

1. If you have not already done so, set up your projects in the Parasoft Test UI.
2. (Optional) if needed, use a localsettings file to override the Parasoft settings specified in the UI. See Configuring Localsettings for details.

cli execution

To run a Code Review scan from the command line interface, use a command such as:

- `parasofttestcli -publishteamserver -config "team://xtest-codereview.properties" -resource "my_resource"`
- `cpptestcli -publishteamserver -config "team://xtest-codereview.properties" -resource "my_resource" -localsettings C:\tmp\localsettings.properties`

The following options are used here:

- **-publishteamserver:** Publishes test results to Team Server.
- **-config:** Specifies Test Configuration For example:

    - -config "builtin://Demo Configuration"
    - -config "Demo Configuration"
    - -config "user://My First Configuration"
    - -config "team://Team Configuration"
    - -config "team://teamconfig.properties"
- **-resource:** Specifies project(s)/file(s) to be tested. For example:
    - -resource "Acme Project"
    - -resource "/MyProject/src/com/acme/MyClassTest.java"
    - -resource "/MyProject/src/com/acme"
    - -resource testedprojects.properties

If you are using a localsettings file to override the settings specified in the UI, you can add `-localsettings`; for example:

- `parasofttestcli -publishteamserver -config "team://xtest-codereview.properties" -resource "my_resource"  -localsettings C:\tmp\localsettings.properties`

Using -config with parasofttestcli if you have multiple Parasoft products installed

If you have multiple Parasoft products installed and want to run tests using parasofttestcli (rather than the product-specific commands such as jtestcli, cpptestcli, etc.), be sure to add the name of the product after -config to specify which product's Test Configuration you want to use.

For example, to use parasofttestcli to run the SOAtest "Code Review" Team Test Configuration, you might use

```
parasofttestcli -config "soatest.team://Code Review" -resource "my_resource"
```

Notifications

After each Code Review scanner execution, the designated reviewers will then be alerted that code is ready for review. The reviewers can perform the review as described in Reviewers - Reviewing Code Modifications.

After the review is completed, the authors can respond as described in Authors - Examining and Responding to Review Comments.

Running Post-Commit Scans with a Pre-Commit

Process

Some teams who submit code for review via a pre-commit process also like to perform a regularly-scheduled post-commit scan in server mode (from the product's command line interface) to:

- Establish a two-tiered notification system where both of the following occur:
    - IDE notifications alert each team member to review tasks. This is achieved by enabling the **Notify me about new or updated reviews every [ ] minutes** option.
    - The server emails summary report and send remainders to reviewers and monitors about pending code review tasks. Emails will be generated if you have the appropriate reporting options set. If you enable the **Generate comprehensive report (includes all scanners)** option, the report will include tasks from all code review scans (pre-commit and post-commit). Otherwise, it will include only tasks from the current post-commit scan. For more details about configuring report generation from command line runs, see Generating Reports.
- Identify any code changes that were committed to source control, but were not submitted for review using the pre-commit procedure.

If your team chooses to use post-commit scans with a pre-commit process, you can:

- **Run a post-commit scan on an empty workspace:** This does not perform any additional scanning or report any additional tasks; it only generates emails related to the submitted "pre-commit" reviews (if the appropriate reporting options are set).
- **Setup a workspace, then run a post-commit scan using -include to "filter in" and rescan only the critical parts of the project that you want scanned:** This finds code changes in critical project areas that were not submitted for review in the pre-commit code review process and generates new tasks for them.
- **Setup a full workspace, and run a post-commit scan on all the code:** This finds any code changes that were not submitted for review in the pre-commit code review process and generates new tasks for them.

Are you trying to ensure that all checked in code has been reviewed?

Many teams want to enforce a policy that requires:

- Every code change must be reviewed
- Developers are not allowed to check-in code without it being reviewed.

One way to achieve this is to set up a hybrid code review process where developers are expected to submit code to reviewers in pre-commit code reviews, then post-commit mode is used to verify that this process is being followed.

The post-commit scan will report a code review task unless the corresponding pre-commit task has been marked as "done". You can customize the timeframe used to determine corresponding pre-commit tasks by changing the settings in the Filter tab's **Pre-commit search range** in days option. By default, the search goes back 7 days from the post-commit scan.

The manager can be set as a monitor. To ensure that all code has been reviewed before each release or build, he can then check that 1) no post-commit tasks are reported to him and 2) the monitor email does not show any pending review items from the pre-commit process.

Be aware that in hybrid code review processes, the scanner will compare changes reported for pre-commit code review with changes committed to source control.

- If the changes are the same, it means that file was reported to review by pre-commit code review and next committed to source control. In this case, the pre-commit process is being followed.
- If the changes are not the same, it means that the file was modified, but not submitted for code review. In this case, the pre-commit process is not being followed.

However, sometimes minor changes that do not require review are introduced into source control—for instance, CVS headers like:

```
/*
* $RCSfile: MyFile.txt,v $
* $Revision: 1.13 $
*
* Comments:
*
* (C) Copyright Parasoft Corporation 1996.  All rights reserved.
* THIS IS UNPUBLISHED PROPRIETARY SOURCE CODE OF Parasoft
* The copyright notice above does not evidence any
* actual or intended publication of such source code.
*
* Revision 1.2  2006/02/03 10:07:28  dan
* class repackaged
*
* Revision 1.1  2005/09/18 09:26:24  mark
* new file
*/
```

In such cases, the Test Configuration for the nightly post-commit scan should specify a regular expression that describes the automatically-generated code. This is entered in the **Post to Pre-Commit matching** option in the Test Configuration's **Code Review> Filters** tab.

For example, to ignore changes in the above CVS header, you would enter `(^ \* .*)|(^ \*$)`

You can also set the Test Configuration to match (and then ignore) other changes that do not need to be reviewed. For example, the team might allow documentation changes to be committed directly to source control (without requiring review).