Configuring Task Assignment and Code Authorship Settings

This topic explains how to configure C++test to compute authorship and assign quality tasks across the team.

Sections include:

About Task Assignment and Authorship

There are typically many different team members working on a development project throughout the SDLC—from developers of application logic, to developers of the Web interface, to QA testers verifying end-to-end processes and transactions.

Using technologies ranging from peer review workflow automation, to static analysis, to unit testing, to runtime error detection, to SOA policy enforcement, to functional testing of individual SOA components as well as end-to-end test scenarios, C++test generates quality tasks for the various team members to perform. These tasks can then be assigned to the appropriate team members based on manual assignments or automatically-detected authorship data.

This task assignment is used to:

- Allow each team member to import only his or her assigned tasks from command line tests into the UI.
- Give each team member a report that contains only his or her assigned tasks.
- Indicate in the manager report which team member is responsible for which tasks.

Developer Task Assignment

For developers, code authorship data can affect how much code is tested and which test results that are shown in each developer's Quality Tasks view.

By changing a Test Configuration's Scope settings, you can configure tests to cover only files or lines that you authored, or only files or lines that you modified after a specified date. Moreover, if the **Test only files/lines authored by [your_username**] Scope option is set for the Test Configuration used for a test, the Quality Tasks view will show only testing tasks from code that you authored and any additional tasks that were reassigned to you. (Error reassignment is discussed in config_task_assgment_code).

QA Task Assignment

For QA, C++test can assign tasks based on source control data in a number of ways:

- If you are going to be statically analyzing source files in your source control system or executing functional tests with test files are stored in your source control system, you can set up C++test to use data from your source control system in order to assign tasks. Static analysis tasks are assigned to the person who introduced them. Test failures are assigned to the person who last worked on the related test.
- If you will be using C++test to automate peer review, review task assignment is defined through the Code Review interface (described in the Code Review section).

If you have added source files to a project in the C++test environment, you can see the team member assigned to a particular line of that source file, as well as information about when it was last modified:

1. Open an editor for the appropriate file.
2. Right-click the line whose author you want to view, then choose **Parasoft> Show Author at Line** from the shortcut menu

Note that if you are not using source control data to calculate task ownership, the message will show modification information for the file (rather than for the specific line selected).

Understanding C++test's Author and Task Assignments

C++test can assign error ownership according to source control data (from supported source control systems), the @author Javadoc tag, xml mapping files (direct file-to-author mappings), and/or the current local user.

If you use multiple sources to determine authorship, authorship priority is determined by reading your settings in the Preference panel's Scope and Authorship page from top to bottom (source control first, @author tags second, a direct mapping file next, current user last). However, if one of the selected options does not determine an author (for instance, the @author tag was selected but the file does not have an @author tag), C++test determines authorship based on the next option selected. If C++test still fails to determine an author, the user is set as "unknown". Likewise, if none of these options is selected, the user is set as "unknown."

If you configure C++test to use both source control and @author tags for authorship computation, source control will be checked whenever possible, and @author will be used for projects and files which are not under source control.

If you choose to use the current local user, the local user will be considered the author and the local modification time will be considered the last modification time.

C++test assigns authorship and tasks as follows:

- If authorship is determined based on source control data, C++test reads source control data to determine who last modified the file/line/method, etc. that is related to the task, then assigns the task to that person.
- If authorship is determined based on @author Javadoc tags, C++test looks for the @author Javadoc tag that is nearest to the code associated with the task, then assigns the task to the author whose username is returned.
    - For method-level tasks, it checks the Javadoc for the method, then for the enclosing class, then for the first class of the file; the first non-null author is returned.
    - For class-level tasks, it checks the Javadoc for the class, then for the first class of the file; the first non-null author is returned.
    - For line level tasks, it checks the Javadoc for the enclosing method, then for the class, then for the first class of the file; the first non-null author is returned.
    - For file and file-level tasks, the first author of the first class of the file is returned
- If file-to-author mappings are specified directly, authorship is assigned according to those settings.
- Otherwise, the local user will be considered responsible for the task and the local modification time will be considered the last modification time.
- When a static analysis violation is suppressed, it is associated with the person who committed the suppression.

To see the author assigned to a particular line of source code, as well as information about when it was last modified:

1. Open an editor for the appropriate file.
2. Right-click the line whose author you want to view, then choose **Parasoft> Show Author at Line** from the shortcut menu

Note that if you are not using source control data to calculate authorship, the message will show modification information for the entire file (rather than for the specific line selected).

Changing How Authorship is Computed

Authorship computation settings can be specified:

- Through the Scope and Authorship preferences page in the C++test GUI.
- Through localsettings.

To use GUI controls to change authorship computation:

1. Choose **Parasoft> Preferences** to open the Preferences dialog.
2. Select the **Parasoft> Scope and Authorship** category in the left pane.
3. Use the available controls to indicate how you want C++test to compute scope and authorship.
    - **Use source control (modification author) to computer scope**: Source control data will be used to compute authorship.
    - **Use file system (xml map) to compute scope**: You will directly specify how you want tasks assigned for particular files or sets of files (for example, you want developer1 to be responsible for one set of files, developer2 to be responsible for another set of files, and so on). See Specifying File-to-Author Mappings for details.
    - **Use file system (current user) to compute scope**: The local user name will be used to compute authorship.
4. Click **OK** to set and save your settings.

For details on how to set authorship settings through localsettings, see Configuring Localsettings.

Specifying File-to-Author Mappings

If you want to directly specify which authors are responsible for which files (rather than have authorship calculated automatically, you can specify file-to-author mappings. These mapping can be automatically configured once, then shared across the team using the auto-configuration process described in C++test Configuration Overview.

To directly specify how you want particular files or sets of files assigned:

1. Indicate that you will be entering mappings directly as follows:
    a. Choose **Parasoft> Preferences** to open the Preferences dialog.
    b. Select the **Scope and Authorship** category in the left pane.
    c. Select **Use file system (xml map) to compute scope**.
2. Enter file-to-author mapping details as follows:
    a. Select the **Scope and Authorship> Authorship Mapping** category in the Preferences dialog.
    b. Specify your mappings in the authorship mapping table. Note that wildcards are allowed; for example:
        - ?oo/src/SomeClass.java - assigns all files whose names starts with any character (except /) and ends with "oo/src/
        - **.cs - assigns all *.cs files in any directory
        - **/src/** - assigns every file whose path has a folder named "src"
        - src/** - assigns all files located in directory "src"
        - src/**/Test* - assigns all files in directory "src" whose name starts with "Test" (e.g., "src/some/other/dir/TestFile.c")

Mapping Order Matters

Place the most general paths at the end of the mapping. For example, /ATM/** path is the last here because it is the most general:
/ATM/unittests/** user 1

/ATM/other/** user 2

/ATM/** user 3

This assigns `unittests` files to user 1, `other` files to user 2, and all other ATM project files to user1. If you reversed the order, all ATM files would be assigned to user 3.

c. Click **Save Changes**.

3. If you're not already sharing these preferences across the team, click **Export** to export the mappings as an XML file, then have team members import the mapping file.

Alternatively, you can direct C++test to use the exported XML file by either:

- **From the GUI** - Specifying the path to that file in the Preferences dialog's **Scope and Authorship> Authorship Mapping** category (using the **Shared File** option).
- **From the command line** - Specifying the path to that file using `scope.xmlmap=true, scope.xmlmap.file=[file],` and `scope.mappings.location` properties (via the `-local-settings` option described in Configuring Localsettings and [Parasoft Product Name] User's Guide> Setup and Testing Fundamentals> Running Tests> Running Tests from the Command Line Interface. Note that this will override any authorship settings specified in the GUI.

A sample XML authorship mapping file follows:

```
<?xml version="1.0" encoding="UTF-8" ?>
<!DOCTYPE authorship (View Source for full doctype...)>
<authorship>
  <!-- assigns all files named: "foo/src/SomeClass.java" to "author1" -->
  <file author="author1" path="foo/src/SomeClass.java" />

  <!-- assigns all files whose names starts with any character (except /) and ends with "oo/
src/SomeClass.cs" to "author2" -->
  <file author="author2" path="?oo/src/SomeClass.cs" />
  <!-- assigns all *.c files in any directory to "author3" -->
  <file author="author3" path="**.c" />
  <!-- assigns every file whose path has a folder named "src" to "author4" -->
  <file author="author4" path="**/src/**" />
  <!-- assigns all files located in directory "src" to "author5" -->
  <file author="author5" path="src/**" />
  <!-- assigns all files in directory "src" whose name starts with "Test" i.e. "src/some/
other/dir/TestFile.java" to "author6" -->
  <file author="author6" path="src/**/Test*" />
</authorship>
```

Generating an XML Mapping File From Source Control

A script can be used to generate an XML mapping file from a source control system. The following sample Perl script demonstrates how to generate an authorship mapping file from CVS. If you are using a different source control system, make the appropriate modifications.

```
#!perl

############################################################
# GetAuthors.pl
# Sample script to generate file --> author mapping.
# Note that this is usually run only once, when you first
# deploy C++test .
#
# This script assumes that you have a list of all the files in the
# project. For example, such a list can be obtained on windows
# by using "dir /s/b *.c" for c files.
############################################################

sub GetAuthor
############################################################
# Gets the author (person who last modified the file) for
# the input file.
############################################################
{
   my $fileName = shift(@_);
   my $author = "DontKnow";
   system qq(cvs log $fileName > cvslog.txt);
   open(TMPFILE, "< cvslog.txt") || die "Could not open cvslog.txt";
   while (<TMPFILE>)
   {
         chop;
          if (m/(^date.*author:)(.*)(;[\s]*state)(.*)/)
          {
              $author = $2;
          }
   }
   close TMPFILE;
   return $author;
}

print "<?xml version=\"1.0\" encoding=\"UTF-8\" ?>\n";
print "<mapping>\n";

############################################################
# AllCFiles.txt has a list of all the c files in the
# project. The following loop iterates through all the
# files and assigns owners to each.
############################################################
open(FILELIST, "<AllCFiles.txt") || die "Could not open AllCSFiles.txt";
while (<FILELIST>)
{
    chop;
    my $fileName = $_;
    my $author = GetAuthor($fileName);
    print "<file path=\"$fileName\" ";
    print "author=\"$author\" />\n";
}

print "</mapping>\n";
```

Specifying Author-to-Author and Author-to-Email Mappings

By default, C++test assumes that each username value it detects (e.g., based on source control data, the system's local user settings, @author tags, direct mapping files, imported scanner.properties files) is the code author's username, and that the related developer's email is [username]@[mail_domain].

However, in some cases, you might want to map the detected username to a different username and/or email address. For example:

- If you want to reassign all of one developer's tasks to another developer (for instance, if User1 has left the group and you want User2 to take care of all the tasks that would be assigned to User1). We will refer to this type of mapping as an "author to author mapping."
- If a developer's username does not match his email address (for instance, the detected username is john but the appropriate email is john_doe@domain.com). We will refer to this type of mapping as an "author to email mapping."

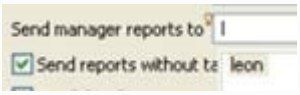
These authorship settings can be automatically configured once, then shared across the team using the auto-configuration process described in C++test Configuration Overview.

If the appropriate authorship settings ARE NOT already set from the auto-configuration process—or if you want to change/override the imported settings—you can map the default user value detected to a different username and/or email address as follows:

1. Choose **Parasoft> Preferences**.
2. Open the **Parasoft> Authors** tab.
3. Ensure that **Use DTP settings** is not enabled.
4. Ensure that **Authors mapping file location** is not enabled.
5. If you want to specify "author to email" mappings:
    a. Got to the **Login / Email / Full name** table.
    b. Click **Add.**
    c. Specify the desired mapping in the table
6. If you want to specify "author to author" mappings:
    a. Go to the **Reassign tasks from this author... To this author** table.
    b. Click **Add**.
    c. Specify the desired mapping in the table.
7. Click the **Apply** button to save your changes.

Specifying Authors in the UI - with DTP Integration

In all the UI locations that allow you to enter a team member's name or email address (e.g., email notifications, reassign tasks, test only files authored by, etc.), you can start typing a name/address, then select one of the autocomplete suggestions.



A warning will alert you if enter a name or email that is not in the DTP database—for example, in case you made a typo.

Specifying Author Mapping in localsettings

You can also specify the author mappings in a localsettings file (from the command line or in the C++test preferences stored on DTP).

To do this:

1. Set `authors.mappings.location=local.`
2. Use `authors.mapping` to specify the appropriate mappings.
3. Use `authors.user` to specify email settings.

For example:

```
authors.mappings.location=local

authors.mapping1=baduser,gooduser

authors.mapping2=brokenuser,fixeduser

authors.mapping3=olduser,newuser

authors.user1=gooduser,gooduser@parasoft.com,Gooduser Stowe

authors.user2=fixeduser,fixeduser@parasoft.com,Fixeduser White

...
```

Determining Case Sensitivity Mode

You can determine whether author values are treated as case sensitive or case insensitive. This case sensitivity setting applies to author values in:

- Code Review
- Quality Tasks
- Reports
- Scope and Authorship
- DTP Tasks
- Test Configurations (e.g., Scope tab > Author options and Code Review tab)

The case sensitivity setting here should match the case sensitivity setting in DTP. By default, DTP is case sensitive. For details on configuring case sensitivity in DTP, see the DTP User's Guide.

Case Sensitivity Details

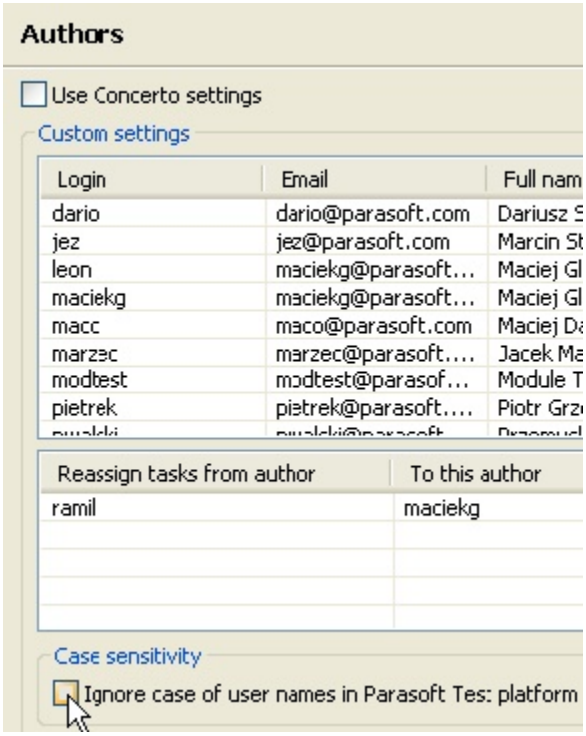
The case sensitivity setting determines whether multiple variations of the same user name will be treated as one user or multiple users, For example:

- Case sensitive: David and DAVID will be treated as two different users.
- Case insensitive: David and DAVID will be treated as one user (the same user).

Case insensitive mode is especially helpful when teams use an external database where the case of the user names is not significant. If the username `david` is stored as `DAVID` in the database and you are in case sensitive mode, tasks imported for `DAVID` will not be matched up to `david`.

Case Sensitivity Configuration

You can set case sensitivity settings in the Preferences panel (**Parasoft> Authors> Ignore case of user names in Parasoft Test platform**).



You can also configure this mode with the localsettings option `authors.ignore.case=true|false`. For details on localsettings options, see Configuring Localsettings.