Collecting Coverage for Web Applications

Export this Guide to PDF

In this guide:

Executive Summary

This guide is intended to help you collect coverage information for applications developed with .NET framework or Java technology.

The primary audience for this user guide are people responsible for ensuring compliance with your organization's policy regarding the application coverage level, including QA Engineers, developers, and build masters.

Prerequisites

We assume that you are familiar with Parasoft technologies and have already deployed and licensed the following products:

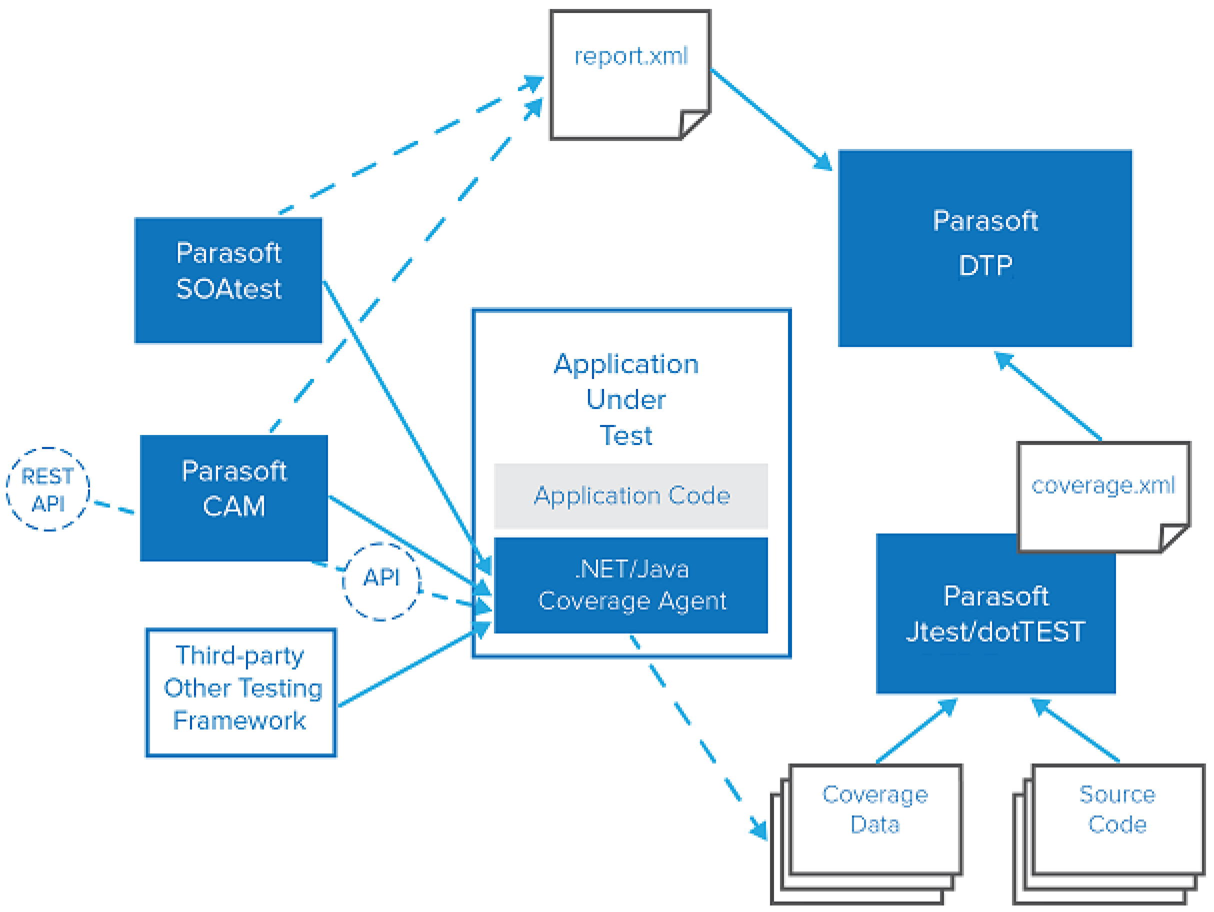
- dotTEST or Jtest 10.4.0 or higher
- Coverage Agent Manager (CAM) or SOAtest
- Parasoft DTP

The following options must be configured in the .properties file where you configure your dotTEST or Jtest settings to ensure that coverage data is correctly displayed on DTP:

- `report.coverage.images` - Specifies a set of tags that are used to create coverage images in DTP. A coverage image is a unique identifier for aggregating coverage data from runs with the same build ID. DTP supports up to three coverage images per report.
- `session.tag` - Specifies a unique identifier for the test run and is used to distinguish different runs on the same build.
- `build.id` - Specifies a build identifier used to label results. It may be unique for each build, but it may also label several test sessions executed during a specified build.

Application Coverage Workflow

1. **Generating the static coverage file.**
   The static coverage file (static_coverage.xml) is generated by Jtest or dotTEST. It is an XML file that contains metadata about user classes, methods, and lines.
2. **Attaching the Coverage Agent to the application under test (AUT).**
   The Coverage Agent ships with Jtest and dotTEST, and allows you to monitor the code being executed when the AUT is running.
3. **Connecting Coverage Agent Manager (CAM) to the Coverage Agent.**
   Create the connection before you start interacting with the AUT.
4. **Performing test sessions with CAM.**
   CAM allows you to mark the beginning and end of test and test sessions while collecting coverage on the running AUT.
5. **Downloading test results and the dynamic (runtime) coverage data when a test session is finished and stopped in CAM.**
   The dynamic coverage data is saved in the runtime_coverage_[timestamp].data file. The test results are stored in the report.xml file.
6. **Uploading the test results to DTP.**
   Data Collector Upload Form allows you to upload the report.xml file to DTP.
7. **Merging the dynamic and static coverage information and send it to DTP.**
   Jtest or dotTEST merges the static coverage data (static_coverage.xml) and dynamic coverage data (runtime_coverage_[timestamp].data) into a coverage.xml file, and sends the coverage.xml file to DTP. If the coverage images, session tags, and build IDs associated with the coverage.xml file and report.xml file match, DTP can properly aggregate, associate, and display the data in a range of reporting mechanisms.



Step 1: Generate the Static Coverage File

The static coverage file must be generated on the build machine that contains the source code. The name of the file is static_coverage.xml.

Generating Static Coverage with dotTEST

Run the following test configuration on the solution:

```
dottestcli.exe -config "builtin://Collect Static Coverage" -solution SOLUTION_PATH
```

The location of the static_coverage.xml file will be printed to the console.

Generating Static Coverage with Jtest

The static coverage file is included in the monitor.zip package generated by the Jtest Plugin for Maven, Gradle, or Ant during the build process.

1. Add the `monitor` task or goal to your build command and execute the command in the AUT's main directory to generate the monitor.zip package. The location of the package will be printed to the console.

   Maven

   ```
   mvn package jtest:monitor
   ```

   Gradle

   ```
   gradle assemble jtest-monitor -I [INSTALL]/integration/gradle/init.gradle
   ```

   Ant

```
ant -lib [INSTALL]/integration/ant/jtest-ant-plugin.jar -listener com.parasoft.Listener jtest-monitor
```

Ant requires all classes to be compiled before the monitor task is executed. Modify your project prior to the build and configure the task to ensure the correct sequence. The following example shows how the target can be configured:

```
<target name="jtest-monitor" depends="compile">
            <jtest:monitor/>
</target>
```

2. Extract the contents of the monitor.zip package to the server machine.

Step 2: Attach the Coverage Agent to the Application Under Test (AUT)

Attaching the Coverage Agent to the AUT allows you enable collecting dynamic (runtime) coverage.

Attaching the Coverage Agent with dotTEST

To attach the Coverage Agent to the AUT, you need to launch the IIS Manager tool shipped with dotTEST on the machine where IIS is installed and the AUT is deployed.

1. Deploy the AUT to the application server.
2. Copy the [DOTTEST_INSTALLATION_DIR]\integration\IIS directory to the machine were IIS is installed and the AUT is deployed.
3. Run a console as Administrator.
4. Invoke the following command to launch the IIS Manager tool:

```
dottest_iismanager.exe
```

5. Go to the following address to check the status of the coverage agent: `http://host:8050/status`. If the Coverage Agent is attached, you should receive the following response:

```
{"session":null,"test":null}
```

Attaching the Coverage Agent with Jtest

To attach the Coverage Agent to the AUT, you need to add Jtest's `-javaagent` VM argument to the startup script of the server where the AUT is deployed.

1. Deploy the AUT to the application server.
2. Extract the contents of the monitor.zip package you generated (see Generating Static Coverage with Jtest) to the server machine. The package contains the agent.sh/agent.bat script.
3. Run the agent.sh/agent.bat script to print the Jtest Java agent VM argument to the console:

```
Jtest Agent VM argument:
-javaagent:"[path to agent dir]\agent.jar"=settings="[path to agent properties file]\agent.properties",
runtimeData="[path to monitor dir]\monitor\ runtime_coverage"
```

4. Add the `-javaagent` argument to the application server's startup script.
5. Restart the server.
6. Go to the following address to check the status of the coverage agent: `http://host:8050/status`. If the Coverage Agent is attached, you should receive the following response:

```
{"test":null,"session":null,"testCase":null}
```

Step 3: Connect CAM or SOAtest to the Coverage Agent

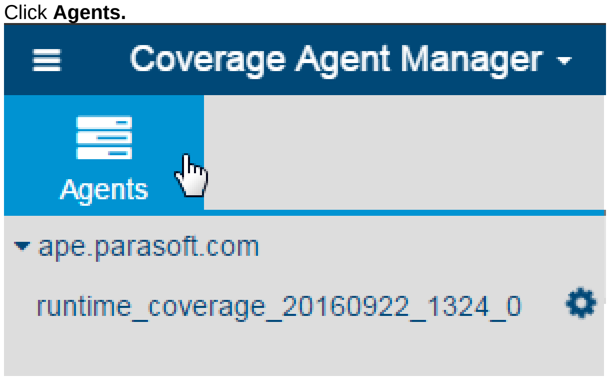
You can connect to either CAM or SOAtest to the Coverage Agent. CAM provides an interface for starting and stopping test sessions during manual test execution. Connecting SOAtest enables you to collect application coverage during automated functional test execution.

Connecting CAM to the Coverage Agent

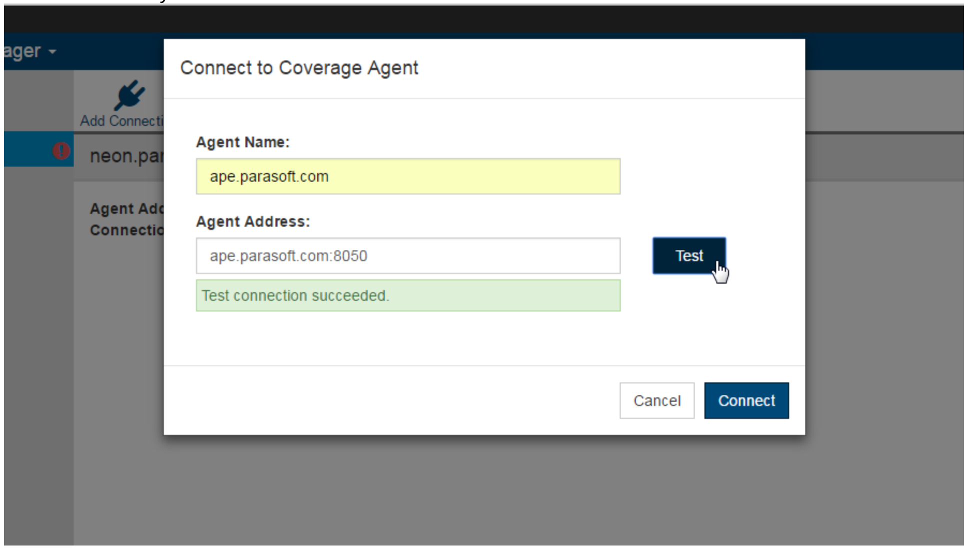
1. Open CAM in a browser:

```
http://[your-Tomcat-host:port]/cam.
```

2. Click **Agents.**



3. Click **Add Connection** to configure the connection with the Coverage Agent.
Agent Name: Any user-friendly string that you want to display in CAM (for example, the name of the application).
Agent Address: The URL of the server where the Coverage Agent is attached to the AUT (see Step 2). The default port number for the Coverage Agent is 8050.
4. Click **Test** to verify the connection.



Connect to Coverage Agent

Agent Name: ape.parasoft.com

Agent Address: ape.parasoft.com:8050 Test

Test connection succeeded.

5. Click **Connect** after successfully testing the connection.

Connecting SOAtest to the Coverage Agent

The connection to the coverage agent is handled through a SOAtest test configuration.

1. Open SOAtest and choose **Parasoft> Test Configurations**.
2. Right-click the **Demo Configuration** in the Built-in folder and choose **Duplicate.** This will create an editable duplicate in the User-defined test configuration folder.
3. Rename the duplicate test configuration and click the **Execution** tab.
4. Click the **Application Coverage** tab and configure the following settings:
    - Enable the **Enable Test Execution** option.
    - Enable the **Collect application coverage** option.
    - Specify the host or IP address of the AUT in the **Coverage agent host** field. The coverage agent should have been attached to the AUT in the previous step.
    - Specify the port number for the coverage agent. The default number is 8050.
    - If you are using multiple-user mode, specify an agent user ID, such as the current SOAtest username. This enables coverage to be associated with a specific user or client. Do not specify an agent user ID if the agent is not in multiple-user mode.
    - Enable the **Report coverage agent connection failures as test failures** option if you want test failures reported when the coverage agent connection fails. If the option is disabled, connection problems will only be reported to the console.
    - Enable the **Retrieve coverage data** and specify the where you want the store the dynamic coverage data in the **Coverage data storage directory** field.
    - Enable **Delete coverage data on retrieval** option if you want the coverage data on the AUT to be deleted as soon as SOAtest retrieves it. We recommend enabling this option in order to prevent the taking up disk space on the coverage agent machine.
5. Click **Apply** to save the changes.
6. (Optional) Save the test configuration to your workspace. This step makes running the configuration on the SOAtest command line more convenient:

    a. Right-click the new test configuration and choose **Export**.
    b. Save the test configuration to the SOAtest workspace.

In the next step, you will execute your tests using the new test configuration from the command line. But in order for DTP to associated the test results with the coverage information, you will need to create a properties file and specify the build ID used to create the static coverage file. The simplest way is to export the properties referenced by the SOAtest desktop.

1. Choose **Parasoft> Preferences** from the SOAtest menu.
2. Click the **share** link on the Parasoft page.
3. Specify a name and location to export the settings file.
4. Enable any settings you want to include in the file and click **OK**.
5. Open the exported file and add the following properties:
    a. `report.coverage.images` - Specifies a set of tags that are used to create coverage images in DTP. A coverage image is a unique identifier for aggregating coverage data from runs with the same build ID. DTP supports up to three coverage images per report.
    b. `session.tag` - Specifies a unique identifier for the test run and is used to distinguish different runs on the same build.
    c. `build.id` - Specifies a build identifier used to label results. It may be unique for each build, but it may also label several test sessions executed during a specified build.
6. Save the changes.

Step 4: Perform Testing Sessions

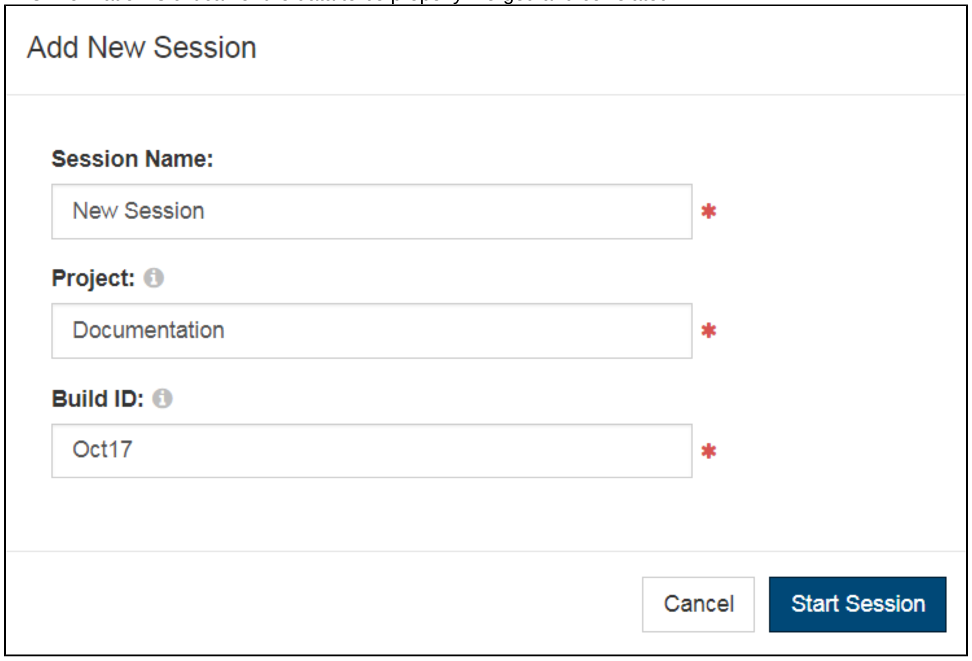
Parasoft provides several ways to execute tests. You can execute tests manually or automate test scenario execution using the command line interfaces. In this guide, we will demonstrate executing manual test scenarios with CAM. We will also describe automated test execution with SOAtest.

Execute Test Scenarios with CAM

To collect coverage data, you need to start a session. Only one tests session can be in progress at a time.

1. Click the **New Session** button, enter the following information when prompted:
    - **Session Name**: the name of the session
    - **Project**: the name of the project in DTP that the results will be associated with
    - **Build ID**: A build identifier for label results. It may be unique for each build but may also label more than one test sessions that were executed during a specified build.

    This information is critical for the data to be properly merged and correlated in DTP.



Add New Session

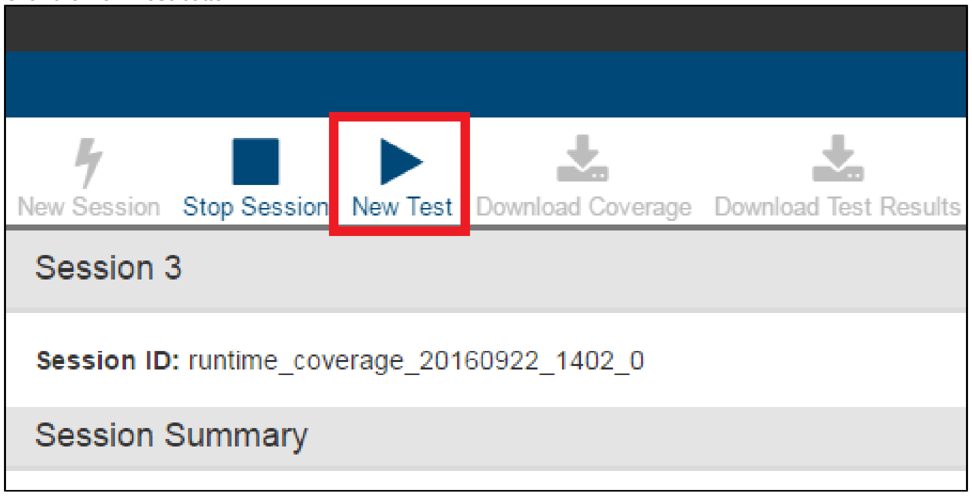
Session Name: New Session

Project: Documentation

Build ID: Oct17

2. Click the **Start Session** button to begin a new test session.

3. Click the **New Test** button.

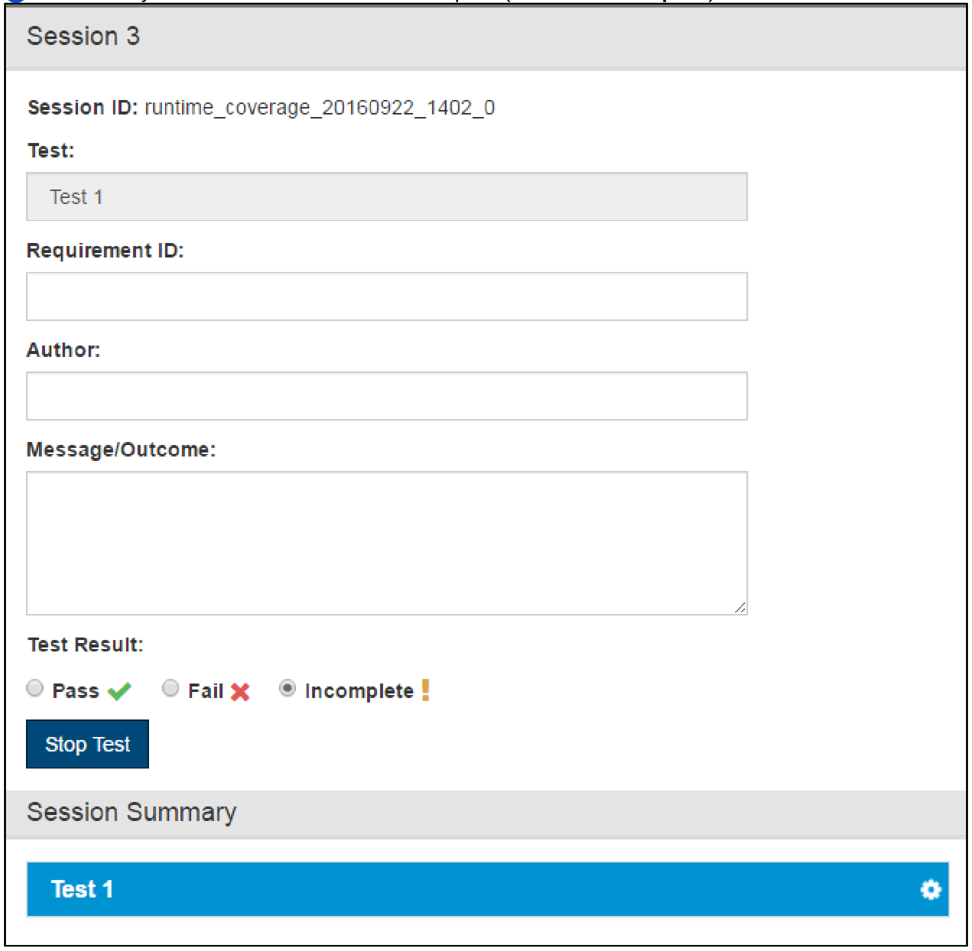


Session 3

Session ID: runtime_coverage_20160922_1402_0

Session Summary

4. Enter a name and click **Start Test** to continue.
5. Perform your manual test on the AUT and enter the data associated with the test.
   Ensure that you select the correct **Test result** option (**Pass**/**Fail**/**Incomplete**).



Session 3

Session ID: runtime_coverage_20160922_1402_0

Test: Test 1

Requirement ID:

Author:

Message/Outcome:

Test Result:

Pass ✓ Fail ✗ [✓ Incomplete !]

Session Summary

Test 1

6. Click **Stop Test** when finished.
7. Continue adding and performing your tests until your scenario is complete.
   Ensure that you stop and start tests in between interacting with the AUT.
8. Click **Stop Session** when you've completed your tests.

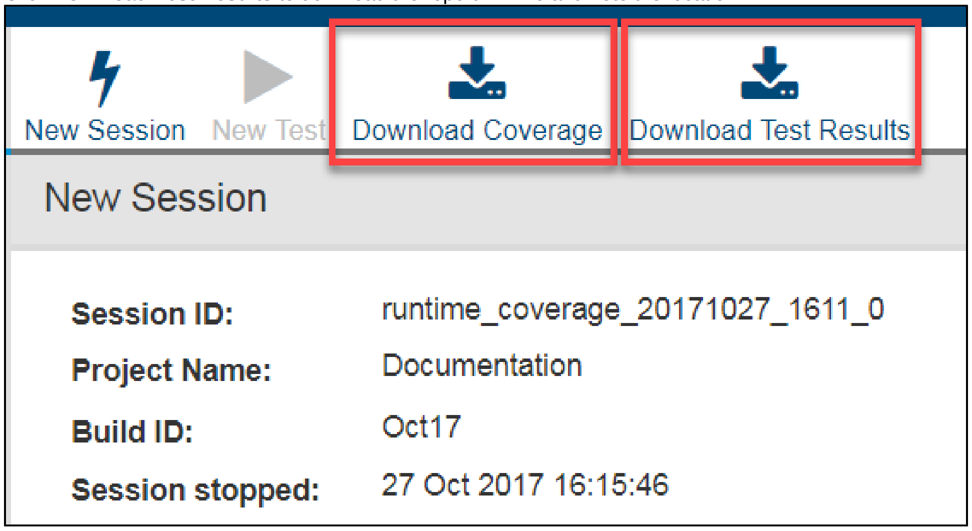
Execute Test Scenarios with SOAtest

Run the soatestcli and using test configuration created in the previous step. You will also need to include the workspace, test assets (.tst files), and properties file. The `-report` flag is optional, but specifying an easy-to-remember location may be helpful as you become familiar with the application coverage workflow:

```
./soatestcli -config "/path/to/your/test-configuration/app-cov.properties" -data "/path/to/your/workspace" -
resource "/path/to/your/tests/your_tests.tst" -localsettings "/path/to/your/localsettings/file/soatest-app-cov.
properties" -report "/directory/to/save/report/"
```

Step 5: (CAM only) Download Coverage and Test Results

1. Click **Download Coverage** to download the runtime_coverage_[timestamp].data file and note the location.
2. Click **Download Test Results** to download the report.xml file and note the location.



Session ID: runtime_coverage_20171027_1611_0
Project Name: Documentation
Build ID: Oct17
Session stopped: 27 Oct 2017 16:15:46

Step 6: Upload the Test Results to DTP

Upload the report.xml file you downloaded from CAM to DTP. If you used SOAtest to execute your tests, the report will be saved in the location specified with the `-report` flag. If you did not specify a location for the report, the file will be saved to the working directory. Uploading test results to DTP allows you to associate coverage data with individual tests and view the information on DTP.

1. Go to **Report Center** in the DTP interface.
2. Click the gear icon and choose **Report Center Settings> Additional Settings> Data Collector Upload Form** (requires admin permissions).
3. Click **Choose File** and browse for the report.xml file you downloaded from CAM.
4. Click the **Upload** button to upload the file to DTP.

Step 7: Merge the Static and Dynamic Coverage Data and Send to DTP

The static and dynamic coverage data must be merged into one coverage.xml file. To merge the coverage data and send the merged information to DTP, run dotTEST or Jtest with the following arguments:

- `-staticcoverage` - Provides the path to the static_coverage.xml generated by dotTEST or Jtest
- `-runtimeCoverage` - Provides the path to the runtime_coverage_[timestamp].data downloaded via CAM or generated by SOAtest. You can provide a path to a folder that contains many .data files from multiple testing sessions.
- `-publish` - sends the merged coverage data to DTP
- (Jtest only) `-config "builtin://Calculate Application Coverage"` - specifies the built-in test configuration required to merge the coverage data

Your command line may resemble the following:

dotTEST

```
dottestcli.exe -staticcoverage [path] -runtimecoverage [path]  -publish
```

Jtest

```
jtestcli -staticcoverage [path] -runtimecoverage [path/dir] -config "builtin://Calculate Application Coverage" -
publish
```

The `-publish` option is required to send the merged coverage data to DTP. Alternatively, you can configure the `report.dtp.publish=true` option in the .properties file where you configure dotTEST or Jtest.

Collecting Application Coverage from Multiple Users

You can collect coverage information for multiple users that are simultaneously accessing the same Java or IIS web application server. This allows QA engineers to perform parallel testing sessions and associate coverage with individual users.

Collecting coverage data from multiple users requires additional configuration steps before you start your testing session. You need to:

1. Configure the Coverage Agent to enable the multiple-user mode; see Configuring the Coverage Agent to Run in the Multiple-User Mode.
2. Specify your User ID for the connection with the Coverage Agent attached to the AUT. This will enable the Coverage Agent to identify and assign coverage information to individual users who are simultaneously interacting with the AUT.
   - If you use SOAtest, provide your User ID on the Application Coverage tab of your test configuration; see Connecting SOAtest to the Coverage Agent.
   - If you use CAM, add your User ID to the HTTP request header of the browser you will use to interact with the tested web application and then provide the same User ID when connecting CAM to the Coverage Agent; see Specifying User ID for Testing with CAM.

Configuring the Coverage Agent to Run in the Multiple-User Mode

Enabling the Multiple-user Mode with dotTEST

You can enable the multi-user mode by modifying the invocation of the dotTEST IIS Manager tool (see Step 2: Attach the Coverage Agent to the Application Under Test (AUT)). Launch the tool with the `-multiuser` switch:

```
dottest_iismanager.exe -multiuser
```

Enabling the Multiple-user Mode with Jtest

You can enable the multiple-user mode by modifying the options in the the agent.properties file, which is included in the monitor.zip package (see Generati ng Static Coverage with Jtest). Open the file and configure the following option:

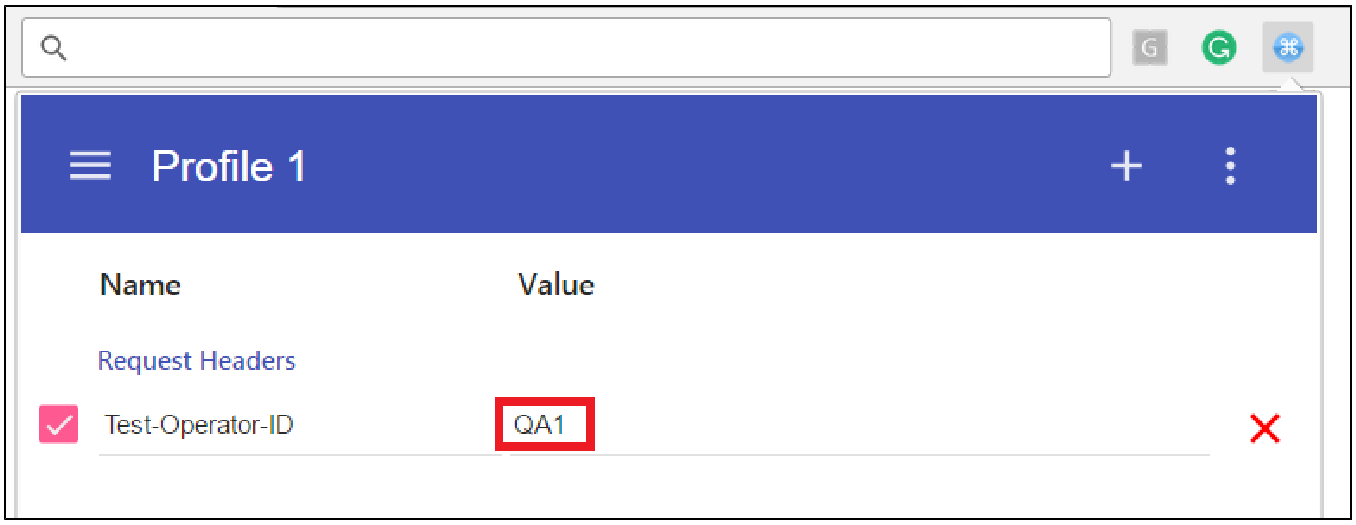
```
jtest.agent.enableMultiuserCoverage=true
```

Specifying User ID for Testing with CAM

Adding the User ID to the HTTP header

Modify the the HTTP request header of your browser by configuring the **Test-Operator-ID** option with a unique User ID.

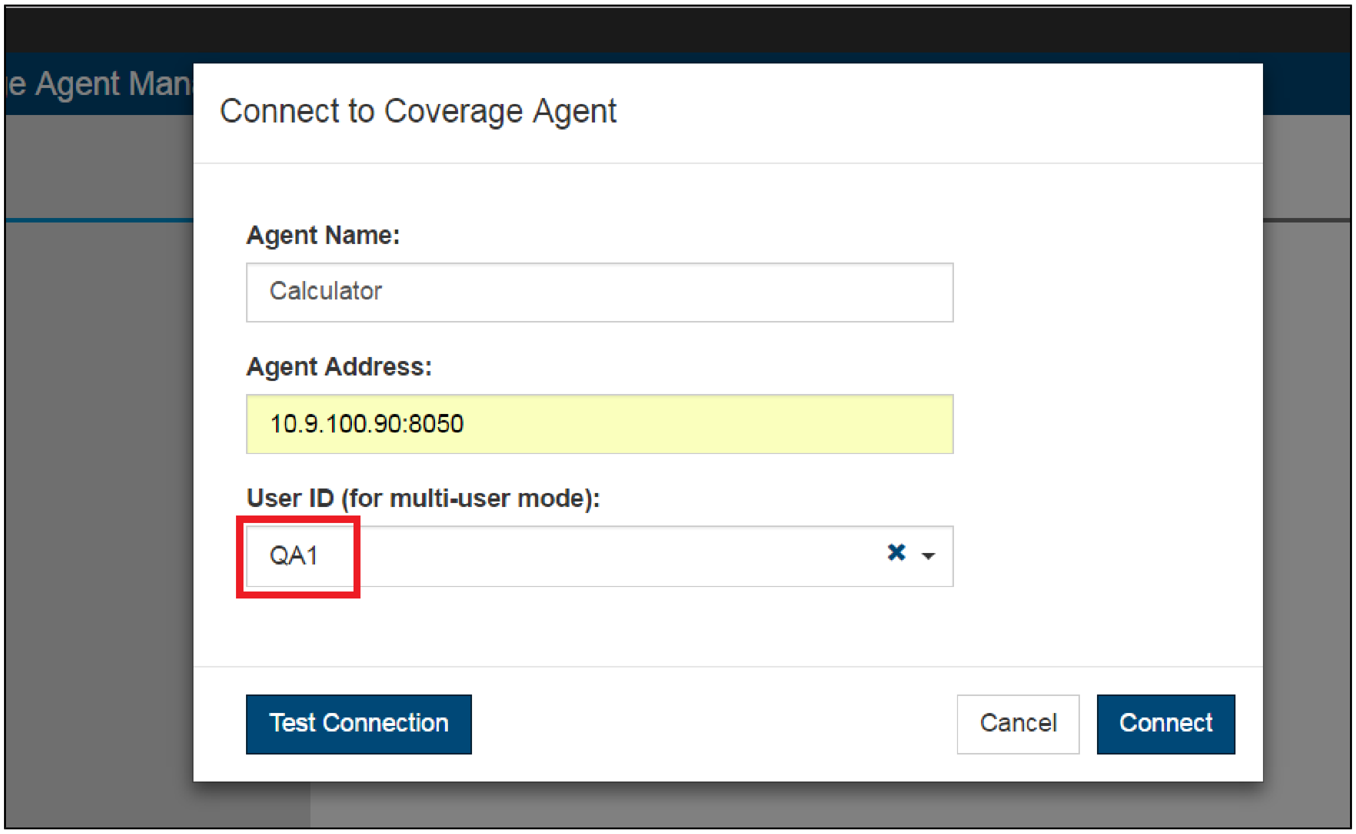
A convenient way to add a Test-Operator-ID is to install one of the browser plugins that allow you to easily modify HTTP headers. The following example shows a Test-Operator-ID specified in Chrome with the ModHeader plugin.



Test-Operator-ID QA1

Connecting CAM to the Coverage Agent in the Multiple-user Mode

Provide your User ID when connecting CAM to the Coverage Agent (see Step 3: Connect CAM or SOAtest to the Coverage Agent). The User ID must be identical to the Test-Operator-ID provided to your browser.



Connect to Coverage Agent

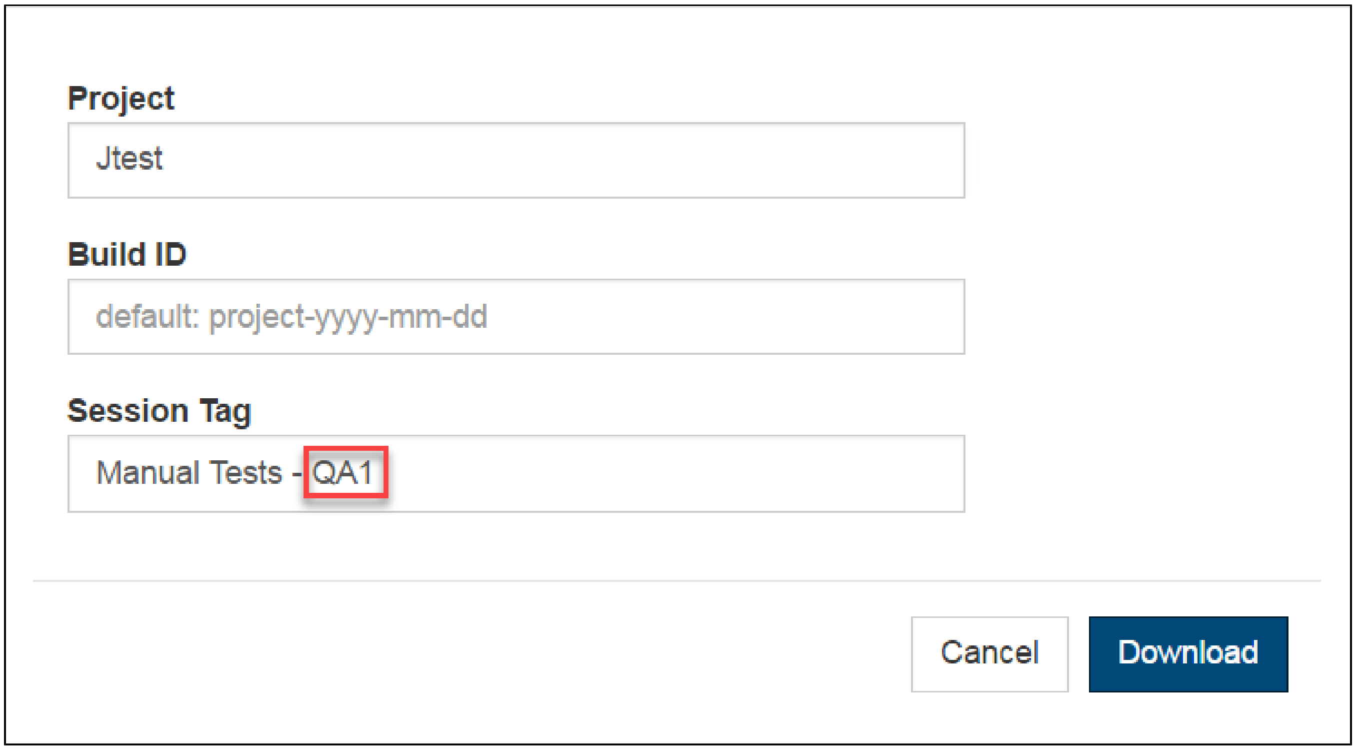
Agent Name: Calculator

Agent Address: 10.9.100.90:8050

User ID (for multi-user mode): QA1

Testing in the Multiple-user Mode

Perform testing sessions, download the results, and merge the coverage data, as described in steps 4-7 above. When downloading the results from CAM, ensure that the **Session Tag** field is completed with a tag unique to each user. We recommend that the session tag include the User ID.


Project

Jtest

Build ID

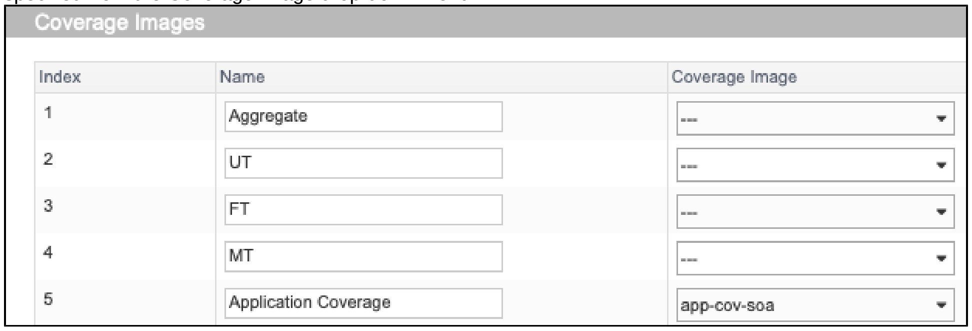
default: project-yyyy-mm-dd

Session Tag

Manual Tests - QA1

Reviewing Application Coverage Results in DTP

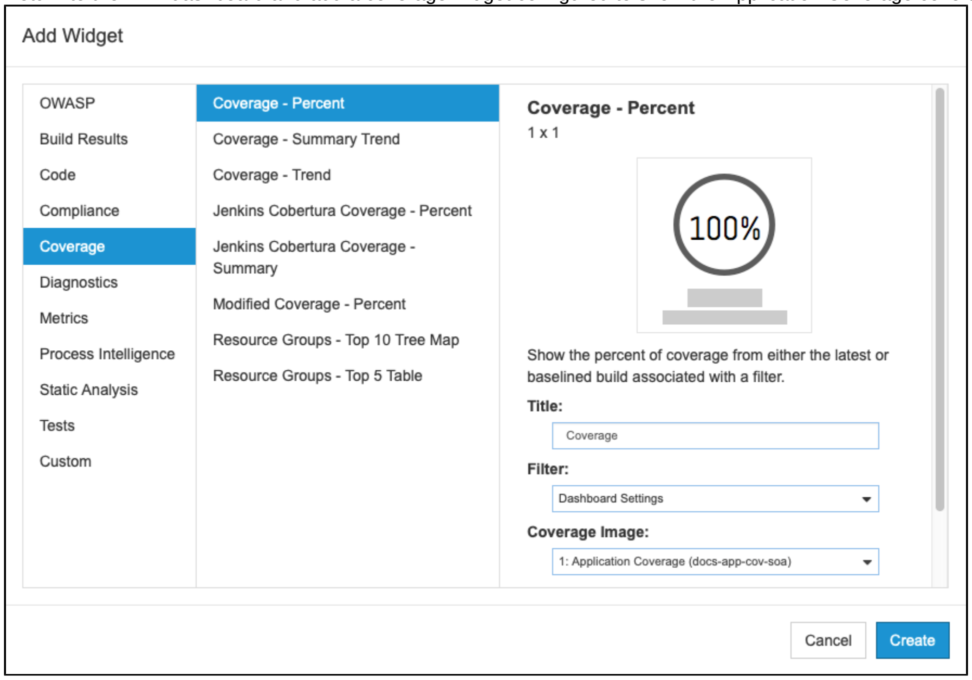
In order to view coverage in the DTP dashboard, you will need to associate the coverage data sent to DTP with the filter.

1. Choose **Report Center Settings** from the DTP settings (gear icon) menu.
2. Click **Filters** and locate your DTP project. By default, a filter is created in DTP for each project. The filter has the same name of the project.
3. Click on the filter and verify that the appropriate run configurations have been added to the filter. A run configuration is a set of metadata about test and analysis executions that enable DTP to make the correct correlations for accurate reporting. If Jtest/dotTEST is configured to report to the project associated with the filter, then the run configurations will automatically be added to the filter by default.
4. In the Coverage Images section, rename one of the image indexes to `Application Coverage` and choose the coverage image tag you specified from the Coverage Image drop-down menu.

Coverage Images

Index	Name	Coverage Image
1	Aggregate	---
2	UT	---
3	FT	---
4	MT	---
5	Application Coverage	app-cov-soa

You can include additional coverage images if you want to view all of your coverage data in one interface. Changes are automatically saved.

5. Return to the DTP dashboard and add a coverage widget configured to show the Application Coverage coverage image.



Coverage - Percent
1 x 1

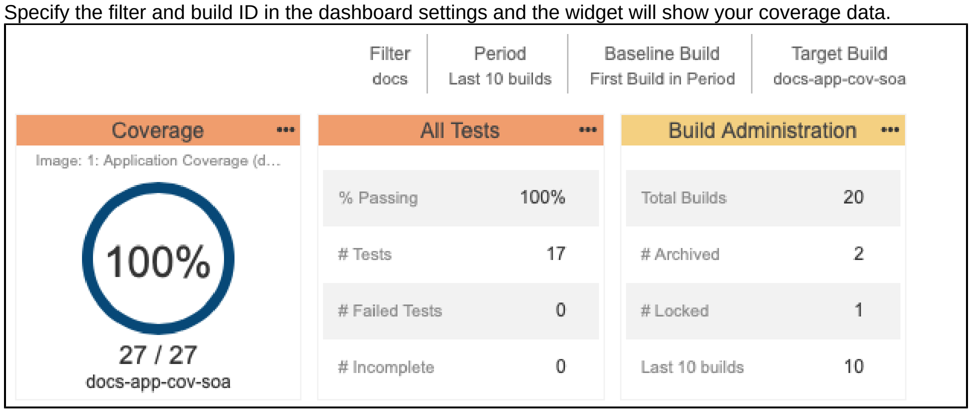
Show the percent of coverage from either the latest or baselined build associated with a filter.

Title: Coverage

Filter: Dashboard Settings

Coverage Image: 1: Application Coverage (docs-app-cov-soa)

6. Specify the filter and build ID in the dashboard settings and the widget will show your coverage data.



Filter	Period	Baseline Build	Target Build
docs	Last 10 builds	First Build in Period	docs-app-cov-soa

Coverage
Image: 1: Application Coverage (d...
100%
27 / 27
docs-app-cov-soa

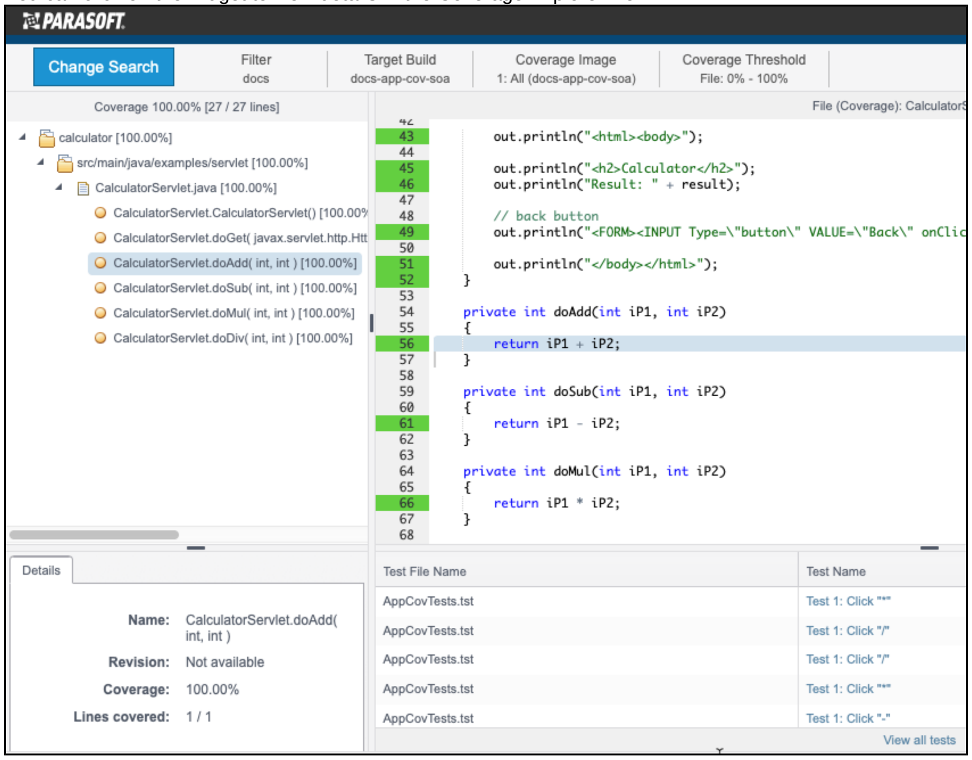
All Tests

% Passing	100%
# Tests	17
# Failed Tests	0
# Incomplete	0

Build Administration

Total Builds	20
# Archived	2
# Locked	1
Last 10 builds	10

7. You can click on the widget to view details in the Coverage Explorer view.



Filter: docs

Target Build: docs-app-cov-soa

Coverage Image: 1: All (docs-app-cov-soa)

Coverage Threshold: File: 0% - 100%

Coverage 100.00% [27 / 27 lines]

- calculator [100.00%]
  - src/main/java/examples/servlet [100.00%]
    - CalculatorServlet.java [100.00%]
      - CalculatorServlet.CalculatorServlet() [100.00%
      - CalculatorServlet.doGet(javax.servlet.http.Htt
      - CalculatorServlet.doAdd(int, int) [100.00%]
      - CalculatorServlet.doSub(int, int) [100.00%]
      - CalculatorServlet.doMul(int, int) [100.00%]
      - CalculatorServlet.doDiv(int, int) [100.00%]

File (Coverage): CalculatorS

```
        out.println("<html><body>");

        out.println("<h2>Calculator</h2>");
        out.println("Result: " + result);

        // back button
        out.println("<FORM><INPUT Type=\"button\" VALUE=\"Back\" onClic

        out.println("</body></html>");
    }

    private int doAdd(int iP1, int iP2)
    {
        return iP1 + iP2;
    }

    private int doSub(int iP1, int iP2)
    {
        return iP1 - iP2;
    }

    private int doMul(int iP1, int iP2)
    {
        return iP1 * iP2;
    }

```

Details

Name:	CalculatorServlet.doAdd(int, int)
Revision:	Not available
Coverage:	100.00%
Lines covered:	1 / 1

Test File Name	Test Name
AppCovTests.tst	Test 1: Click "*"
AppCovTests.tst	Test 1: Click "/"
AppCovTests.tst	Test 1: Click "/"
AppCovTests.tst	Test 1: Click "*"
AppCovTests.tst	Test 1: Click "-"

View all tests