DTP Concepts

Parasoft DTP is a browser-based application that collects, processes, and reports analytics associated with software development. Software quality activities, such as static analysis, flow analysis, unit testing, etc., are executed by Parasoft code analysis and test execution tools, such as C/C++test, dotTEST, Jtest, and SOAtest, and reported to DTP. The data is processed in DTP and made available to developers and testers through a DTP interface, such as dashboard widgets. Development and testing leads can also review and prioritize findings that can be downloaded to team member's IDEs.

Although this documentation covers the DTP functionality, it is often necessary to discuss C/C++test, dotTEST, Jtest, and SOAtest functionality in relation to DTP processes. This section covers important concepts that apply to DTP and the code analysis and test execution tools and that are critical for understanding how our solution functions.

In this section:

- Run
- Run Configuration
- Filter (DTP Server)
- Project (DTP Server)
- Test Configuration
- Session Tag
- Session Tags and Run Configurations
- Build
- Coverage Tag
- Finding

Run

A run in DTP is a single execution of the Parasoft tool. It is equivalent to an XML report produced by the code analysis and test execution tool. A run is uniquely defined with the following attributes:

- Time of run
- Build ID
- Coverage Tag
- Tool (C/C++test, dotTEST, Jtest, SOAtest)
- DTP Project
- Test Configuration
- Session Tag
- Machine Name
- Machine User

Static Analysis Examples

Each of the following static analysis executions are counted as separate runs:

- Code analysis on project A using the "Recommended Rule" test configuration with machine A.
- Code analysis on project A/B using the "Recommended Rule" test configuration with machine A.
- Code analysis on project A using the "Find Duplicated Code" test configuration with machine A.
- Code analysis on project A using the "Recommended Rule" test configuration with machine B.

Each execution produces an XML report that is sent to DTP.

Run Configuration

A Run Configuration represents a of series of runs executed by the same code analysis and execution tool. Project, Test Configuration, and Session Tag are grouped into the same Run Configuration. A Run Configuration is uniquely defined with the following attributes:

- Tool (C/C++test, dotTEST, Jtest, SOAtest)
- DTP Project
- Test Configuration
- Session Tag

Filter (DTP Server)

A filter represents a group of run configurations. Filters enable multiple users working on an application to static analysis results from different projects within the application. Run Configurations must be properly set up use filters.

Static Analysis Example

User A only needs static analysis results for the "core" project. User B only needs results for the "API" project. User C needs a combination of results.

Three different filters can be set up to accommodate each users' needs. The filter for user A would contain a single Run Configuration for "core." The filter for user B would contain a single Run Configuration for "API". The filter for user C would contain both Run Configurations.

Project (DTP Server)

A project is the name of an on-going software development project.

Example

A company with three products may set up an individual project in DTP for each product.

Properties (or Settings)

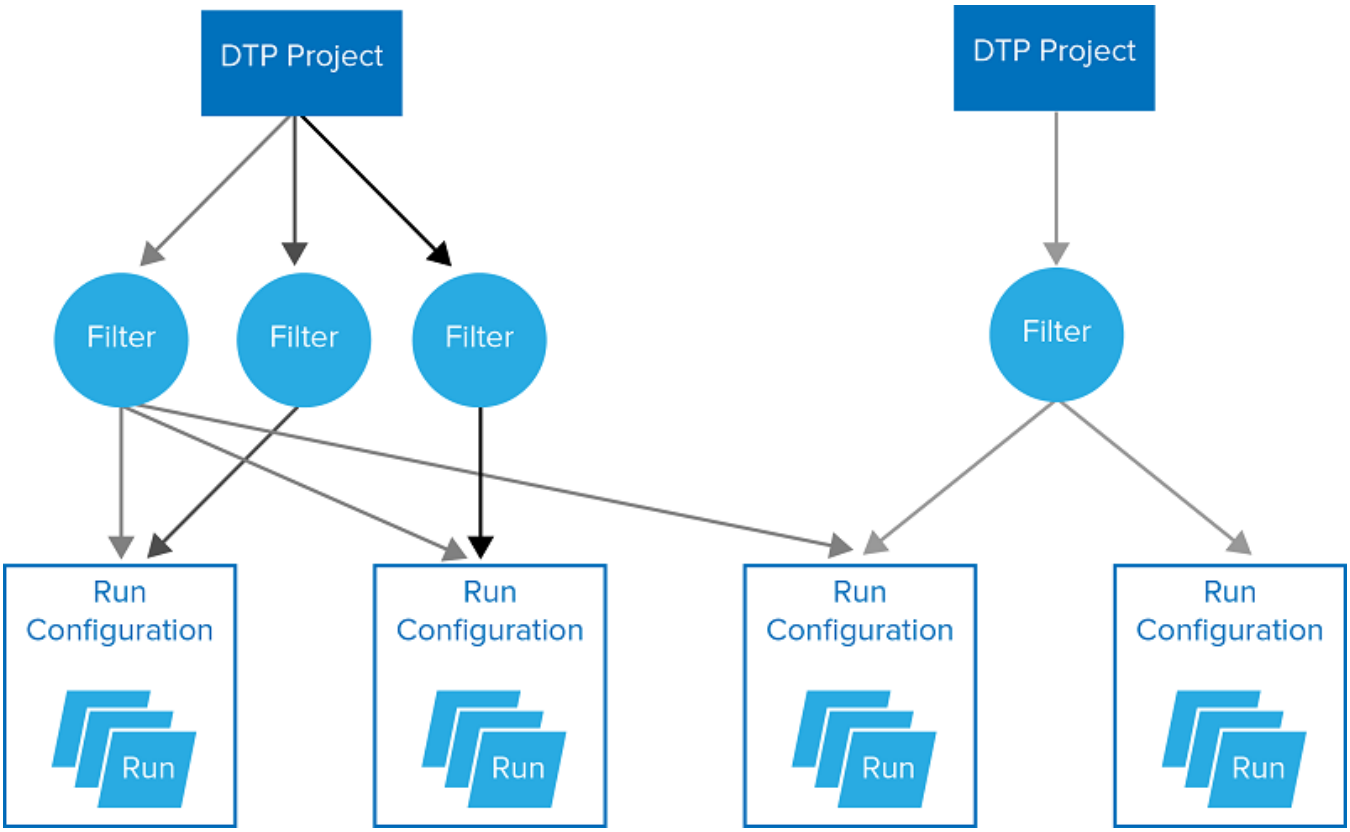
Parasoft tools are configured by setting properties in "properties" or "settings" files. These plain text files have a .properties file extension. Parasoft tools send data to DTP for specific projects identified with the `dtp.project` setting in the .properties file. Refer to the tool documentation for more information about the .properties file.

Default Behavior

If not specified in the .properties file, data sent into DTP is collected into the Default Project.

Example Project Filter and Data Structure

The following chart shows the relationship between DTP projects, filters, run configurations, and runs.



Test Configuration

A Test Configuration represents the settings used by C/C++test, dotTEST, Jtest, or SOAtest during the analysis.

Examples

- Static Analysis Use Case: Use the “Recommended Rules” Test Configuration to analyze code against the set of rules specified in the Test Configuration.
- Unit Test Use Case: Use the “Unit Tests” Test Configuration to execute unit test data according to the Test Configuration.
- Metrics Case: Use the “Metrics” Test Configuration to collect metrics data according to the Test Configuration.

See Test Configurations for details.

Session Tag

A Session Tag represents a unique identifier for the run and is used to distinguish similar runs. The tag is defined with the `session.tag` setting in the tool's .properties file. Refer to the tool documentation for more information about the .properties file.

Default Behavior

If not specified in the tools' .properties file, the following attributes are used:

```
${scontrol_branch}-${exec_env}
```

The `${scontrol_branch}` variable points to the name of the branch.

The `${exec_env}` variable points to the execution environment.

Examples

The following tags could be used when running static analysis on “head” and “release” branches in a 64-bit Linux environment:

- For “head” branch: `session.tag = head-linux_x86_64`
- For “release” branch: `session.tag = release-linux_x86_64`

The following tags could be used when running unit tests in Linux and Windows environments:

- For Linux environment: `session.tag = linux_x86_64`
- For Windows environment: `session.tag = windows_x86_64`
- Use a variable to capture either environment: `session.tag=${exec_env}`

When running static analysis on large codebase, you break up the execution into two smaller executions rather than executing static analysis once on the entire codebase. To distinguish the two runs, you will have to manually set the session.tag.

- For “core” codebase: `session.tag=${scontrol_branch}-core`
- For “API” codebase: `session.tag=${scontrol_branch}-API`

Session Tags and Run Configurations

Using different session tags means that runs with different session tags will be grouped under different Run Configurations. In the first example above, two session tags are used to distinguish similar runs on two different branches. In this case, two different run configurations are set.

If two runs on different branches have the same session tag, (run by the same tool, with the same DTP Project and same Test Configuration), DTP will not differentiate the runs and will interpret them as two runs of the same Run Configuration. In this case, the second run will be considered the newer run. We strongly recommend against using the session tag in this way.

By properly leveraging and combining session tags, Run Configurations, and filters, you are able to see results for a single series of runs (i.e., Run Configuration) or combine results from multiple series of runs.

Build

A build is a unique identifier used to group a set of runs across multiple Run Configurations. Builds are contained in all XML reports, but their primary use is for aggregating data from multiple dynamic analysis executions (e.g., unit test, functional test, manual test) and coverage runs.

A group of reports identified with the same build represent the results from a group of tests against the same build of the application.

The build ID is defined with the `build.id` setting in the tool's .properties file. Refer to the tool documentation for more information about the .properties file.

Default behavior

If not specified in the tool's .properties file, the following attributes are used:

```
${dtp_project}-date_of_run
```

The `${dtp_project}` variable points to the value specified by the `dtp.project` setting. See Project (DTP Server).

Example

See the example in Coverage Tag.

Coverage Tag

The coverage tag is a unique identifier used to aggregate coverage data from runs with the same build ID. Using a different set of coverage tags for individual runs enables you, for example, to present coverage in a single DTP Dashboard from different types of tests (automated unit or functional tests, manual test sessions). The coverage can be viewed separately per practice, as well as aggregated across all testing practices.

The coverage tag is defined with the `report.coverage.images` setting in the tool's .properties file. Refer to the tool documentation for more information about the .properties file.

Default behavior

If not specified in the tool's .properties file, the following attributes are used:

```
${dtp_project}
```

Example: Running Unit Tests and Measuring Coverage

In an application with 2 million lines of code, you have divided your suite of unit tests into three sets. You have configured Run Configurations and filters so that the sets of unit tests are grouped under three different Run Configurations.

DTP will properly aggregate the unit test and coverage results if the build ID and coverage tag for each run are the same. This will enable you to view the correct data in DTP.

While the runs occur on the same date and are associated with the same DTP Project, the build ID for the runs will be the same by default. As a result, we recommend using the default behavior of the build ID for most nightly test infrastructures.

Finding

A "finding" in Parasoft is an individual output from a code analysis run that has been tagged with one or more properties. When Parasoft code analysis tools check code, each output is automatically associated with a set of properties, such as analyzed file, code authorship, analysis rule, etc., in the report. You can configure DTP to automatically attach additional properties, such as priority, action to take, etc., or manually review the data to determine how to process the findings. Findings are most commonly static analysis violations, but can include other types of analysis, such as metrics, that take properties.