

DTP Concepts

The focus of this documentation is on the server functionality of DTP, which includes the collection, processing, and reporting of analytics associated with application development. Software quality activities, such as static analysis, flow analysis, unit testing, etc., are executed by the DTP Engines. This structure should not, however, be thought of as a server/client relationship between DTP and the DTP Engines, though. Rather, the DTP Server and DTP Engines can function independently, but when fully integrated they create a seamless workflow for developers to efficiently detect and remediate software defects, as well as facilitate a continuous process for preventing the defects from ever happening again.

This documentation covers the DTP server functionality, but it is often necessary to discuss the functionality in association with the DTP Engines for C/C++, Java, or .NET. This section covers important concepts that apply to DTP and the DTP Engines and that are critical for understanding how our solution functions.

At a high level, the following concepts are critical to understanding how DTP and DTP Engines interact to form a complete development testing solution.

In this section:

- [Run \(DTP Server\)](#)
- [Run Configuration \(DTP Server\)](#)
- [Filter \(DTP Server\)](#)
- [Project \(DTP Server\)](#)
- [Example Project Filter and Data Structure](#)
- [Session Tag](#)
- [Build](#)
- [Coverage Tag](#)

Run (DTP Server)

A run in DTP Server represents a single execution of the Engine. It is equivalent to an XML report produced by the Engine.

Example: Static Analysis Use Case

Suppose you have a code base of 2 million lines for a product called "Code Parser." The code is structured into two projects, "core" and "API." If you run static analysis on "core" in one execution and static analysis on "API" in another execution the result is two runs. Each execution produces an XML report that is sent to DTP server.

Technical Details

A run is uniquely defined with the following attributes:

- Time of run
- Build ID
- Coverage Tag
- Engine
- DTP Project
- Test Configuration
- Session Tag
- Machine Name
- Machine User

Run Configuration (DTP Server)

A Run Configuration in DTP Server represents a series of runs. Runs with the same Engine, DTP Project, Test Configuration, and Session Tag are grouped into the same Run Configuration.

Technical Details

A Run Configuration is uniquely defined with the following attributes:

- Engine
- DTP Project
- Test Configuration
- Session Tag

Filter (DTP Server)

A filter represents a group of run configurations.

Example: Static Analysis Use Case

Suppose there are three different users working on an application. 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.

Run Configurations must be properly set up to properly set up filters. In addition to the DTP Engine that executes the analysis, three additional attributes (DTP Project, Test Configuration, and Session Tag) determine the uniqueness of a Run Configuration.

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.

Engine Settings (.properties file)

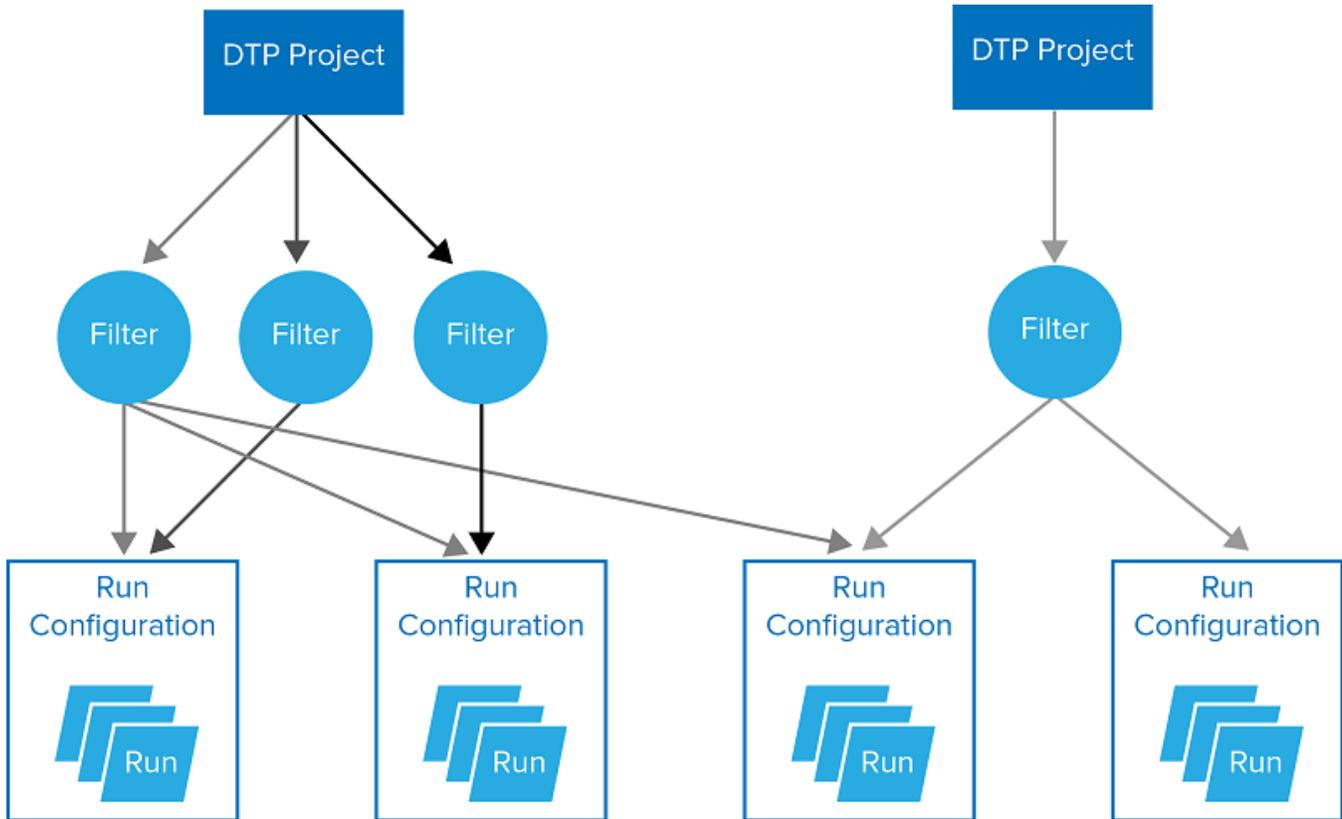
The project is defined with the `dtp.project` setting in the engine's `.properties` file. Refer to the DTP Engines documentation for more information about the `.properties` file.

Default Behavior

If not specified in the DTP Engine `.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 the Engine 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.

Session Tag

A Session Tag represents a unique identifier for the run and is used to distinguish similar runs.

Engine Settings (.properties file)

The session tag is defined with the `session.tag` setting in the engine's `.properties` file. Refer to the DTP Engines documentation for more information about the `.properties` file.

Default Behavior

If not specified in the DTP Engine `.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 Engine, 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. Although builds are contained in all XML reports, its primary use is for aggregating data from multiple dynamic analysis executions (e.g., unit test, functional test, manual test) and coverage runs.

The premise is that a group of reports identified with the same build represent the results from a group of tests against the same build of the application.

Engine Settings (.properties file)

The session tag is defined with the `build.id` setting in the engine's `.properties` file. Refer to the DTP Engines documentation for more information about the `.properties` file.

Default behavior

If not specified in the DTP Engine `.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.

Engine Settings (.properties file)

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

Default behavior

If not specified in the DTP Engine `.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.