Key Performance Indicator

This slice calculates a "KPI" (METRIC.KPIIF) score by applying predefined weighted rule profiles to a given development project. The weighted rule profiles allow you to define custom sets of rules and apply weights to each rule individually. When this slice is invoked, it counts the number of rule violations for each file in the project; for each rule, it multiplies the violation count by the specified rule weight in the profile. If a rule does not have a defined weight in the profile, the count for that rule is not gathered. The sum of the weighted counts is divided by the logical lines of code (METRIC.NOLLOCIF) in the given file to yield a KPI score.

Contact your Parasoft representative to download and license this artifact. This artifact also ships with the Security Compliance Pack for DTP 5.4.2, which includes configurations for calculating KPIs associated with security compliance standards.

In this section:

Requirements

- DTP and DTP Enterprise Pack 5.4.2.
- The code analysis tool must be configured to report the Number of Logical Lines in Files metric (METRIC.NOLLOCIF) to the DTP. The tool must also be configured with the correct filter and build ID. See DTP Concepts for additional information about filter and builds, as well as the code analysis tool for information on how to configure these settings.
- DTP must contain a build with static analysis data. Metrics will be reported to DTP in the latest build with static analysis data.

Added Components

Four example profiles (described below) are added when the KPI slice is installed.

Installation

1. See Downloading and Installing Artifacts for instructions on installing artifacts.
2. Choose **Import> Library> Process Intelligence> Key Performance Indicator> Key Performance Indicator** from the actions menu in Extension Designer to import the artifact into a service.
3. Click **Deploy**.

Upgrade Notes

Uninstall the previous version of the artifact, including models and profiles, if you are upgrading to KPI 2.1.

Starting with 2.1, you can define metric IDs for each profile. Associating profiles with specific metric IDs enables you to view KPIs for multiple metrics without overwriting results.

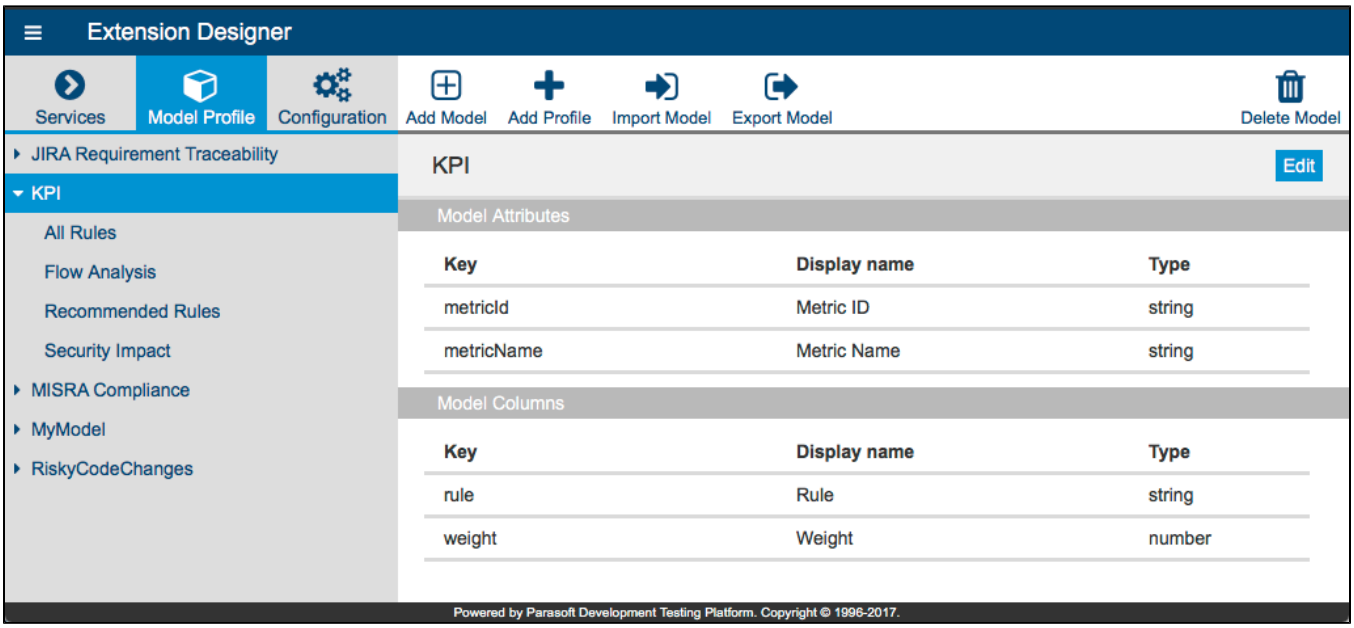
If you altered the default profile shipped with the previous version or created custom profiles, export the profile(s) and add the following attributes (also see Exporting and Importing Profiles):

- metricId: METRIC.KPIIF
- metircName: KPI in File

If you change the default values, we recommend setting the metric ID prefix to `METRIC.KPI.<profile name without spaces>`. Use a concise name for the metric (maximum 30 characters).

Configuration

The slice uses the KPI model profile type to calculate a KPI score. The profile is determined from the “profile” query parameter sent in the widget request. Several example profiles are installed with the slice and can be viewed in the Model Profile tab.



See Working with Model Profiles for additional information.

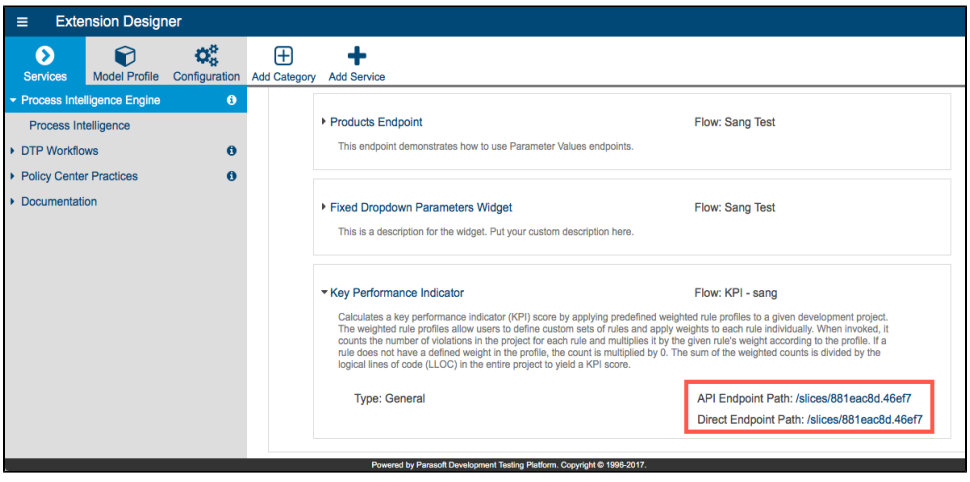
The filter ID is passed by the widget as a query parameter. It determines which DTP filter the profile should be computed against. The following example profiles are installed with the distribution to ease the configuration of the model profiles:

All Rules	This profile assigns a weight to all Jtest static analysis rules equal to 100/severity (e.g., severity 1 violations are weighted a value of 100, severity 2 violations are weighted a value of 50, and so on.)
Recommended Rules	This profile assigns a weight to Jtest static analysis rules that are active in the Recommended Rules test configuration. The weights are calculated in the same fashion as the All Rules profile.
Flow Analysis	This profile assigns a weight to Jtest static analysis rules that are active in the Flow Analysis test configuration. The weights are mapped in the same fashion as the All Rules profile.

Invoking the Calculation

The KPI slice is a long running slice. It will only be computed when invoked by a third-party, ideally as part of a nightly job.

1. Click on the service category on the Services tab and drill down to the endpoint. You can use the API or direct endpoint path (see Service Category Page for additional information).



2. Copy the endpoint and send a REST request to it with the required parameters. Use the API endpoint path if available (see Working with Flows for additional information). The following table describes the required parameters:

filterId	The filter id for the project that the calculations will be performed on.

profile	Profile name with the rules and weights to use for the calculations.
buildId	The build id for which the calculations will be performed on. If no build id is provided, this parameter defaults to the latest build.

Example Invocation

You could run the following command to invoke the slice and run the calculation:

```
curl "http://localhost:8321/slices/a37fbfb6.59b87?filterId=71&profile=All%20Rules"
```

If successful, you will receive a response such as the following:

```
{{success: {title: "KPI", message: "Calculation has successfully started for filter 'Parabank-v3' using profile
'Security Impact'."}}
```


Use HTTP(S)-compliance Parameters

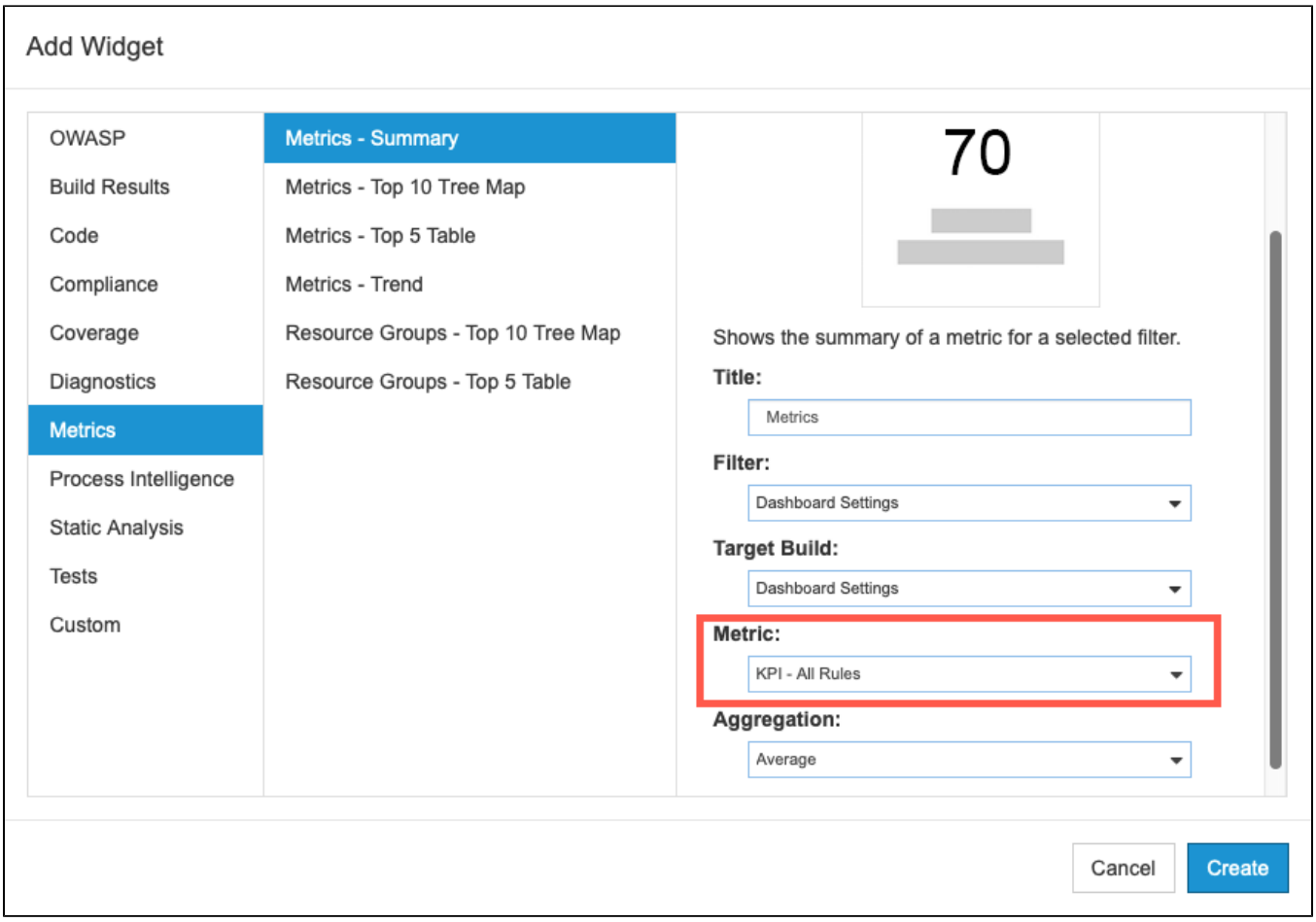
When adding your parameters, be sure to properly encode the string parameter values if they contain spaces, +, /, or any other characters that are not allowed in the HTTP(S) protocol.

The following parameter, for example, is not allowed and will prevent the artifact from functioning correctly: `buildId=c++test`. The encoded parameter would be `buildId=C%2D%2Dtest`.

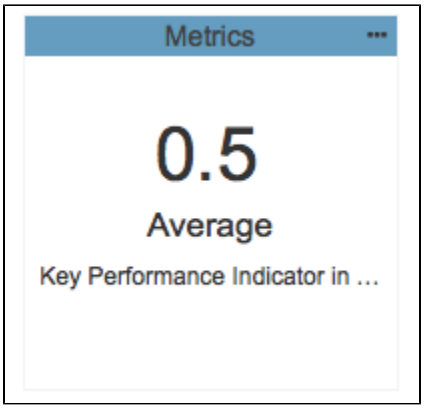
You can use an encoding tool (e.g., http://www.url-encode-decode.com) to help you properly encode parameters to be compliant with the standard.

Viewing the Data in DTP

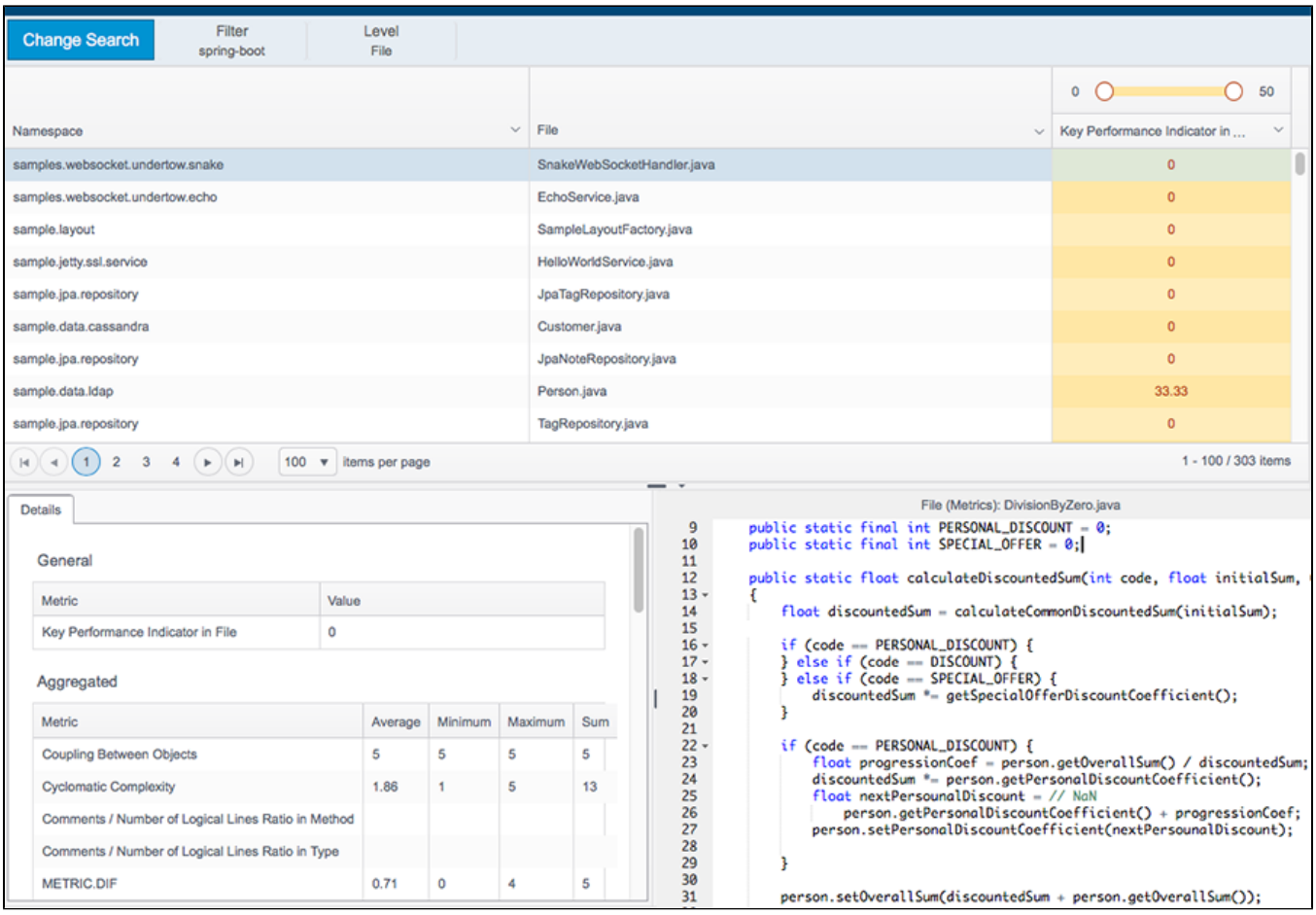
After computation has completed and the KPI metric has been reported back to DTP, you can add a Metrics widget to your DTP dashboard and choose **Key Performance Indicator** from the Metric drop-down menu. See Adding Widgets for additional information.



The widget will display the metric according to your specifications.



Click on the widget to view details in the Metrics Explorer.


Change Search
Filter
spring-boot
Level
File
0
50
Namespace
File
Key Performance Indicator in ...
samples.websocket.undertow.snake
SnakeWebSocketHandler.java
0
samples.websocket.undertow.echo
EchoService.java
0
sample.layout
SampleLayoutFactory.java
0
sample.jetty.ssl.service
HelloWorldService.java
0
sample.jpa.repository
JpaTagRepository.java
0
sample.data.cassandra
Customer.java
0
sample.jpa.repository
JpaNoteRepository.java
0
sample.data.ldap
Person.java
33.33
sample.jpa.repository
TagRepository.java
0
1 2 3 4
100
items per page
1 - 100 / 303 items
Details
General
Metric
Value
Key Performance Indicator in File
0
Aggregated
Metric
Average
Minimum
Maximum
Sum
Coupling Between Objects
5
5
5
5
Cyclomatic Complexity
1.86
1
5
13
Comments / Number of Logical Lines Ratio in Method
Comments / Number of Logical Lines Ratio in Type
METRIC.DIF
0.71
0
4
5
File (Metrics): DivisionByZero.java
9 public static final int PERSONAL_DISCOUNT = 0;
10 public static final int SPECIAL_OFFER = 0;
11
12 public static float calculateDiscountedSum(int code, float initialSum,
13 {
14 float discountedSum = calculateCommonDiscountedSum(initialSum);
15
16 if (code == PERSONAL_DISCOUNT) {
17 } else if (code == DISCOUNT) {
18 } else if (code == SPECIAL_OFFER) {
19 discountedSum *= getSpecialOfferDiscountCoefficient();
20 }
21
22 if (code == PERSONAL_DISCOUNT) {
23 float progressionCoef = person.getOverallSum() / discountedSum;
24 discountedSum *= person.getPersonalDiscountCoefficient();
25 float nextPersonalDiscount = // NaN
26 person.getPersonalDiscountCoefficient() + progressionCoef;
27 person.setPersonalDiscountCoefficient(nextPersonalDiscount);
28
29 }
30
31 person.setOverallSum(discountedSum + person.getOverallSum());