VECTOR

User's Guide
VectorCAST/Analytics

VectorCAST 2021

New editions of this guide incorporate all material added or changed since the previous edition. Update packages may be used between editions. The manual printing date changes when a new edition is printed. The contents and format of this manual are subject to change without notice.

Generated: 9/22/2021, 8:31 PM

Rev: 3a4ade2

Part Number: VectorCAST/Analytics User's Guide for VectorCAST 2021

TABLE OF CONTENTS

Introduction to Analytics

VectorCAST/Analytics

VectorCAST/Analytics provides a web-based dashboard view of software code quality and testing completeness metrics, making it easy to understand the current state of quality and testing completeness for a software project. This critical intelligence allows all stakeholders to make decisions about release readiness and process improvement.

VectorCAST/Analytics features:

> **Real-Time Code Quality Metrics** - Provides quantifiable data on test run vs. tests needed, release readiness, risk areas, and hot spot identification.

> **Technical Debt Identification** - Identifies data on the key components of technical debt such as: code complexity, comment density, and testing completeness.

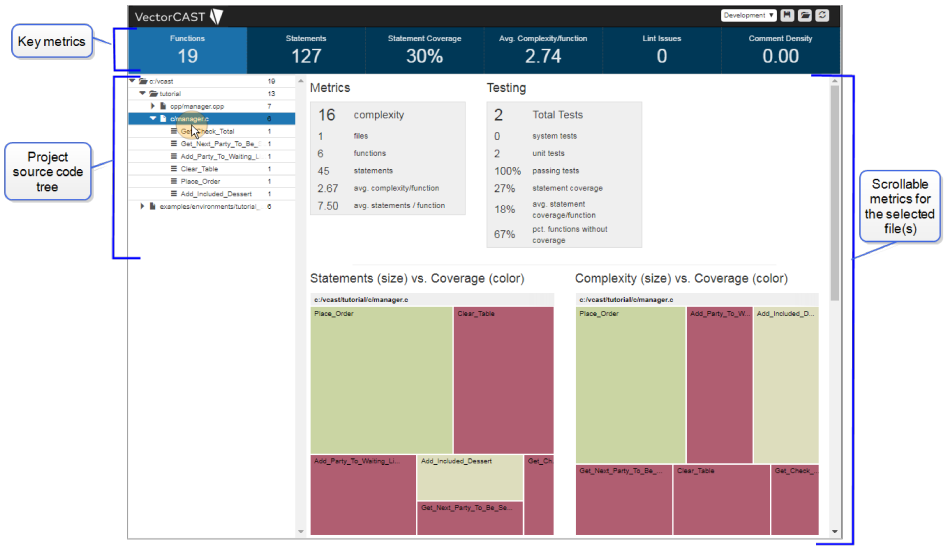
> **Test Case Quality** - Reports on the quality of test cases with metrics such as: tests with expected values but no requirements, number of requirements tested, and tests with expected values.

> **Customization** - Allows customization of calculated metrics, as well as data presentation using a variety of built-in graphs and tables.

> **Extendable Data Connectors** - Includes built-in data connectors for VectorCAST tools and is easily extended to support any third-party data sources.

VectorCAST/Analytics works by providing user-configurable data connectors that allow key metrics such as static analysis errors, code complexity, code coverage and testing completeness to be captured from VectorCAST or third-party tools. These base metrics can be combined into compound metrics to identify hot spots in the code, such as functions with high complexity and low coverage.

Key metrics are shown in tables and treemaps, offering an initial quality assessment to identify high value activities to improving code quality. In a treemap view, where code coverage controls the box color and code complexity controls the box size, users quickly view where they should invest testing and refactoring resources to get the best return on investment. Big red boxes imply highly complex functions that are poorly tested.



Quick Start

Getting Started

This Quick Start chapter is intended to get you started quickly with the basic features of VectorCAST/Analytics. Use it for a quick reference.

Before you start: Ensure that VectorCAST is installed and that the environment variable `VECTORCAST_DIR` is set to the installation directory. Refer to the *Interactive Tutorials* for detailed installation instructions.

The default VectorCAST/Analytics configuration supports all VectorCAST tools. Simply point the VectorCAST Analytics server at any VectorCAST/Manage testing project or VectorCAST/Cover coverage project and the default dashboard displays key metrics in an easy-to-understand layout.

For the purposes of this demonstration, we will create an Enterprise Testing Project using the Enterprise Unit Testing example included with VectorCAST.

Create a VectorCAST Project

Enterprise Testing is a Test Automation Framework that sits on top of VectorCAST/C++ or VectorCAST/Ada test environments and allows test design, execution, and reporting to be distributed across the enterprise. The VectorCAST project supports a variety of work flows allowing for team collaboration, testing of multiple configurations, change-based testing, and massively parallel testing.

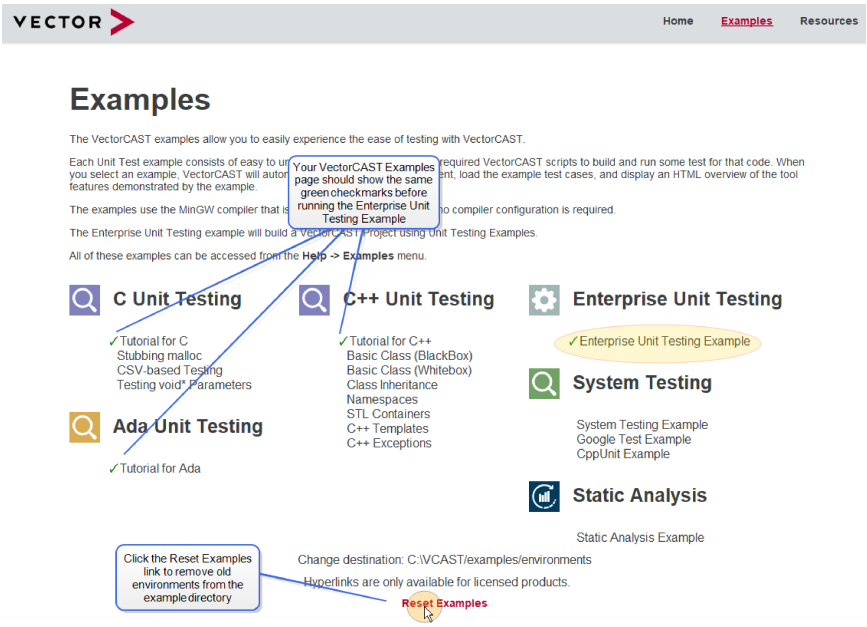
Enterprise Testing can import existing VectorCAST/C++ and VectorCAST/Ada test environments, or be used to create new environments. In this section, you will take existing VectorCAST environments and import them into a VectorCAST Project.

Before you start: The basic Enterprise Unit Testing example is built using any Unit Testing Environments you created previously. If there are no existing environments, the example will create a new unit test environment using the Tutorial for C.

For our example, we will first reset the examples on the Examples page and then import the C, C++, and Ada Unit Testing environments.

To set up our VectorCAST project, go to the VectorCAST Examples page and select the **Reset Examples** link located at the bottom of the page.This removes all of the example environments from your build directory, and gives us a clean directory.

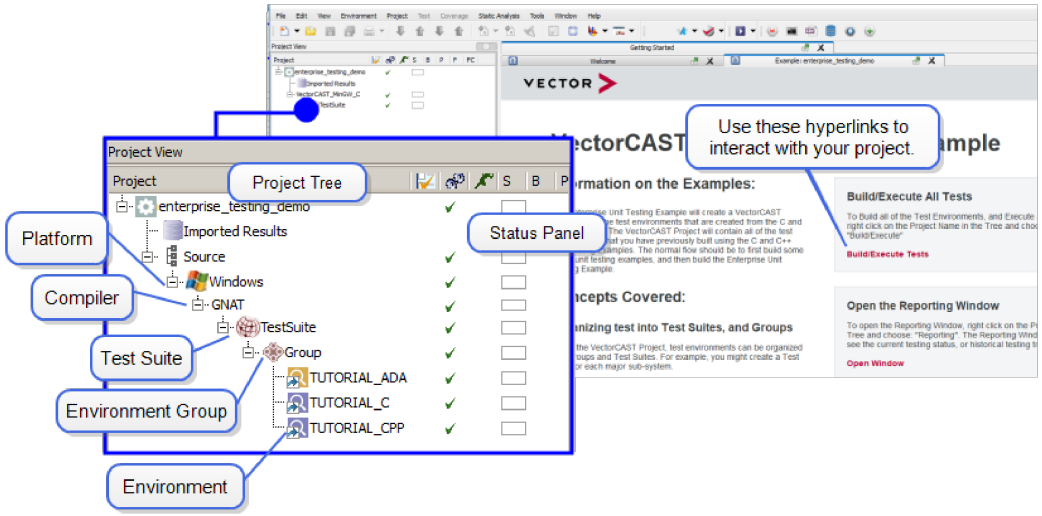
Next we will create our unit environments. Click to run the example Tutorial for C located under the C Unit Testing column. A green check mark is displayed next to it when the build is complete. Return to the VectorCAST Examples page and click on the Tutorial for Ada and the Tutorial for C++ examples so that each also displays a green check mark:



Finally, we create our VectorCAST Project. Navigate to the VectorCAST Examples page and under the Enterprise Unit Testing column, select **Enterprise Unit Testing Example**. Alternatively, from the Menu Bar, you can select **Help =>Example Environments =>Enterprise Testing =>Enterprise Unit Testing Example**. VectorCAST will automatically build the VectorCAST Project using the Unit Testing environments you created previously.

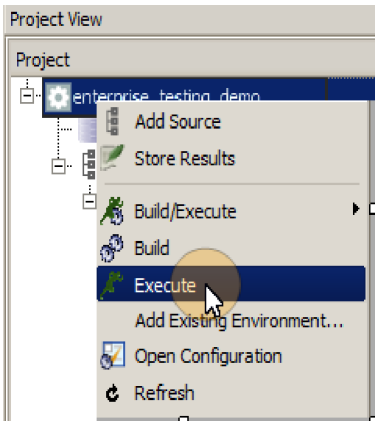
Once the project completes building, a VectorCAST Enterprise Testing Example summary page is provided for you in the MDI Window. Refer to this page to learn more about basic VectorCAST project concepts, and use the provided hyperlinks on the right to interact with your project.

In the Project Tree you will now see the enterprise_testing_demo project displayed. Note that a Test Suite has been created which contains each of the environments. Expand the Project Tree to see the individual Environments. Next, you will execute all of the tests.



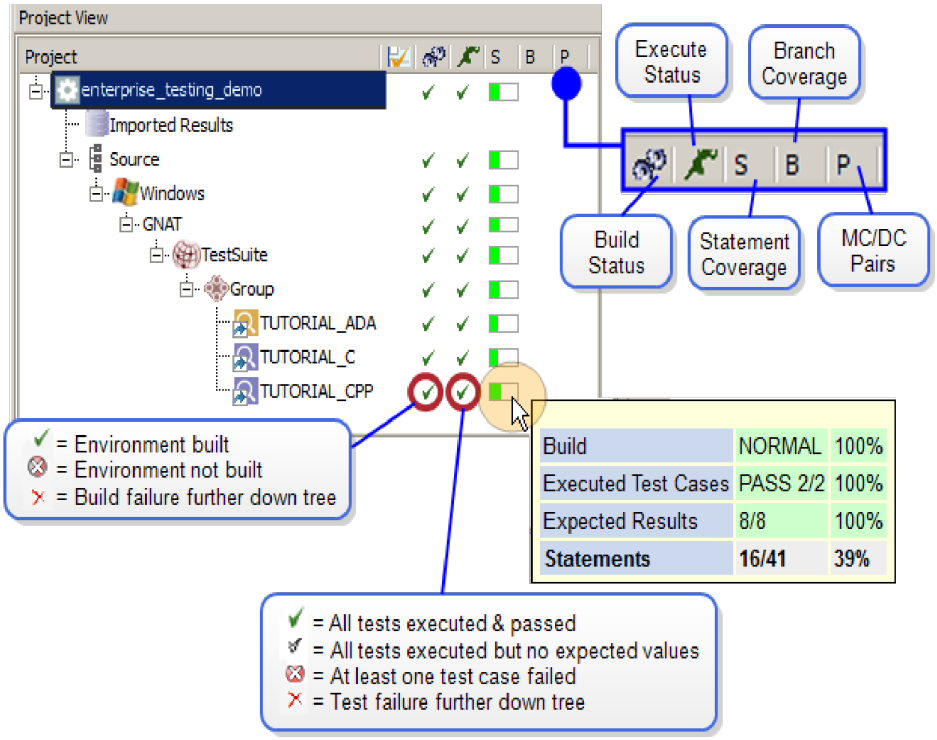
Execute All Tests

To execute all of the tests in the enterprise_testing_demo project, right-click on the Project Name (enterprise_testing_demo) and select **Execute** from the context menu.



You can follow the execute process in the Manage Status viewer which opens in the MDI Window. As test cases are executed data is stored in a SQL database and used to generate reports showing testing status and trends, making it easy to analyze regression trends.

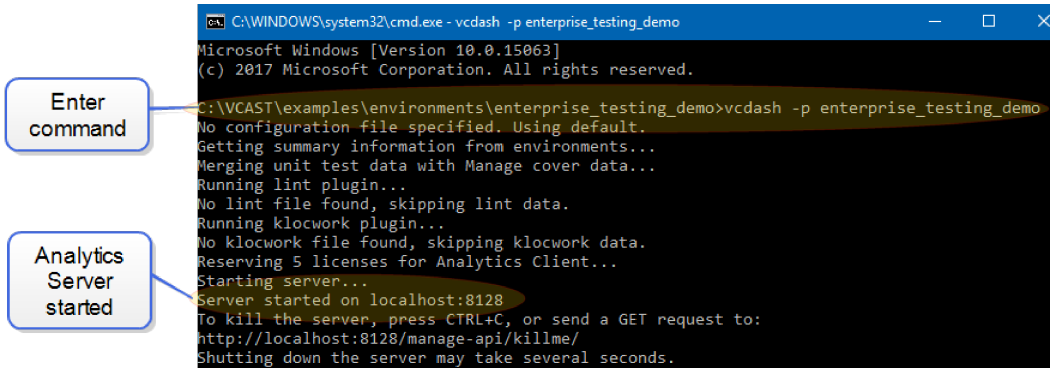
The Status Panel updates to display testing status. On the status panel you will see status for the Environment Build, Test Execution and Statement Coverage. Hover over the Statement Coverage bar to see a pop-up of the Build and Coverage details.



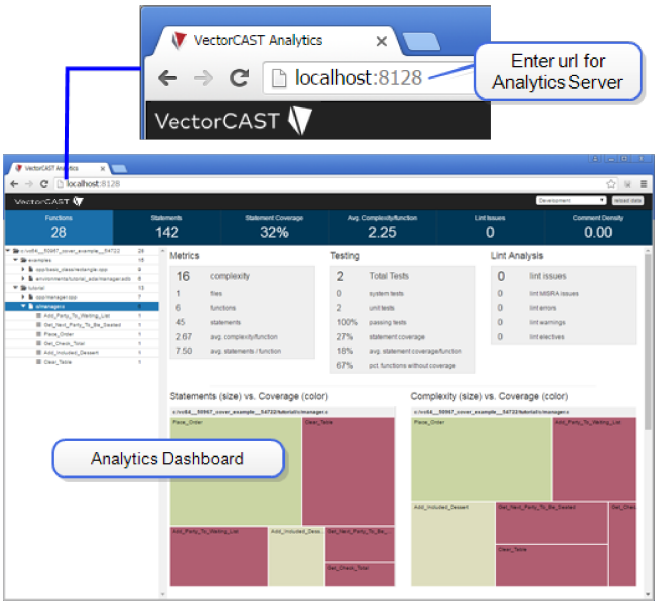
Start Analytics From VectorCAST

To launch the Analytics Server, open a DOS command prompt by selecting the Command Prompt icon from the Toolbar. From the command line, enter:

```
%VECTORCAST_DIR%/vcdash -p enterprise_testing_demo
```



To open the Analytics Dashboard, first open a web browser and enter the web address: **localhost:8128**. The dashboard opens in the web browser.

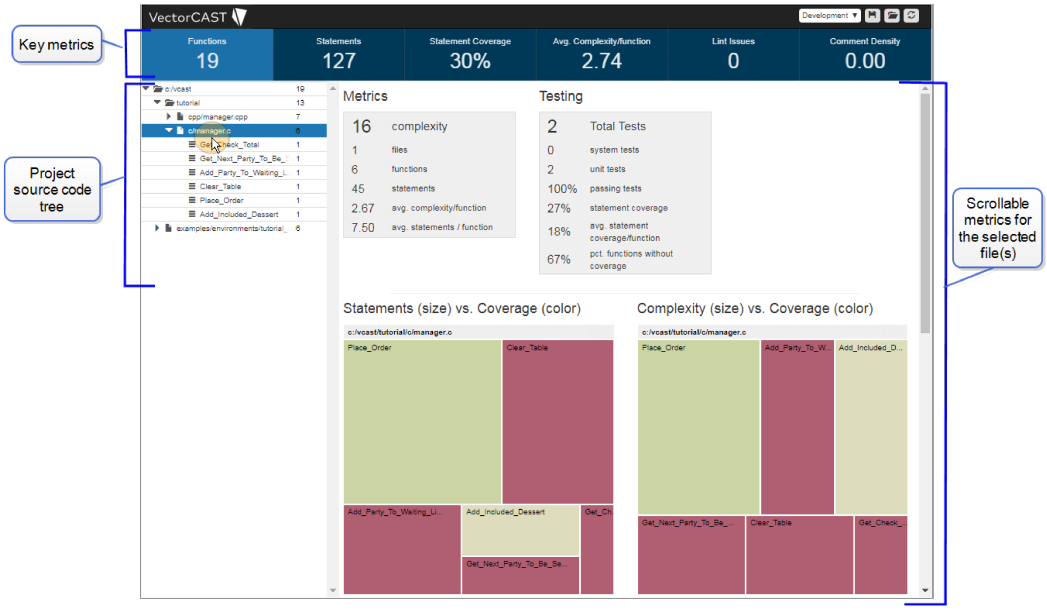


Understanding the Analytics Dashboard

For the purposes of our discussion, we have selected to view the metrics for the source file **`c/manager.c`** by clicking on the link in the Project Source Code Tree.

The Analytics Dashboard is composed of three main areas:

> Key Metrics
> Project Source Code Tree
> Metrics for selected source file(s)

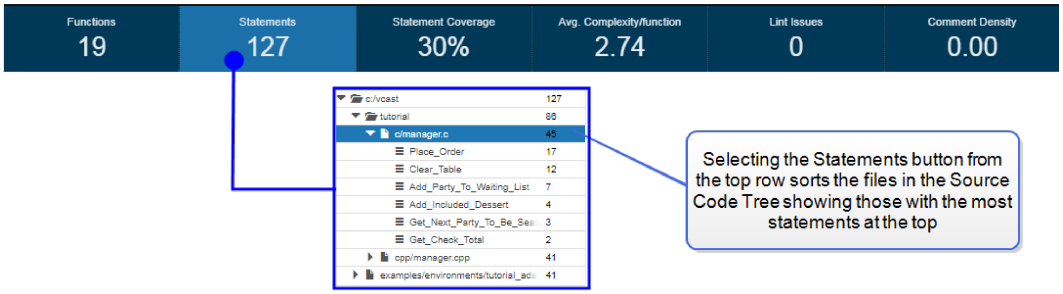


Key Metrics

Key Metrics are project-wide metrics displayed in the top bar of the dashboard, providing an at-a-glance view of the size, complexity and testing completeness of the project. Metrics include:

> Functions - total number of functions in the project
> Statements - total number of statements in the project
> Statement Coverage - percentage of statements covered in the project
> Avg. Complexity/Function - the average code complexity, or V(g), per function
> Static Analysis Issues - total number of static analysis issues in the project
> Comment Density - the percentage of comments to effective lines of code

The Key Metrics are functional buttons which control the sorting and display of the project's Source Code Tree. In our example, the **Statements** button is selected, and the source code files are listed in the Tree showing those having the greatest number of statements at the top.

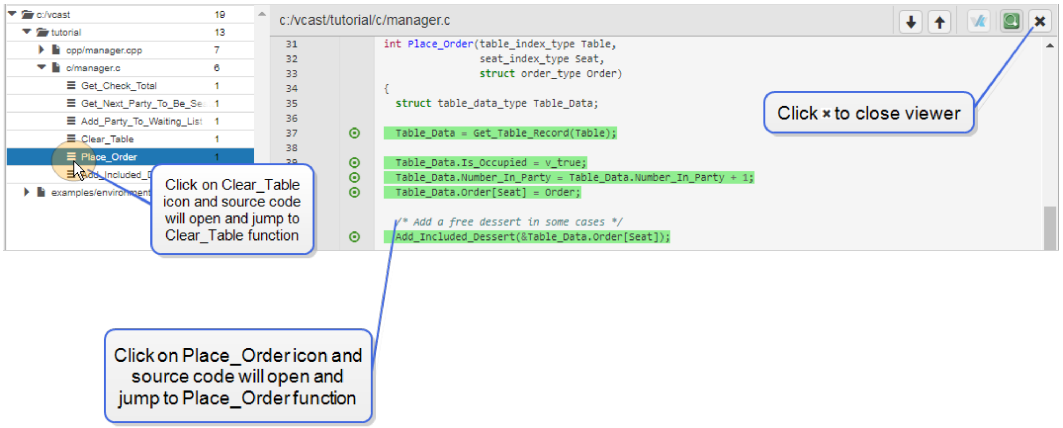


Source Code Tree

The Source Code Tree lists the source files and functions associated with the project. The selections made in the Source Code Tree control the set of functions used to calculate the metrics displayed in the dashboard. Metrics can be displayed for the entire project, or all the way down to metrics for an individual function.

The data displayed in the right column of the Source Code Tree is associated with the selected Key Metric. Select the **Statement Coverage** button, and note that the right column then shows the percentage of statement coverage achieved, listed from lowest to highest percentage.

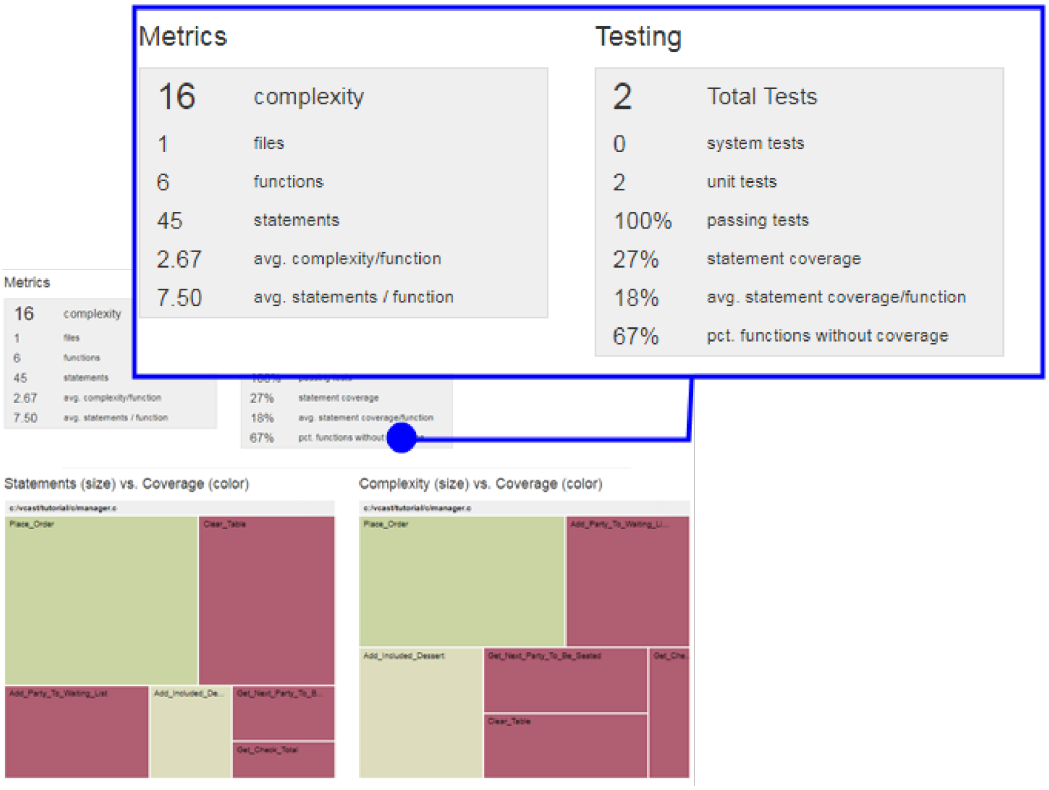
Note that the Source Code Tree uses an icon beside a file name and an icon beside a function name. Clicking on a file name icon will open the file's source code in a Viewer. Clicking on a function name icon will open the file's source code in a Viewer and will jump to the function's location in the code.



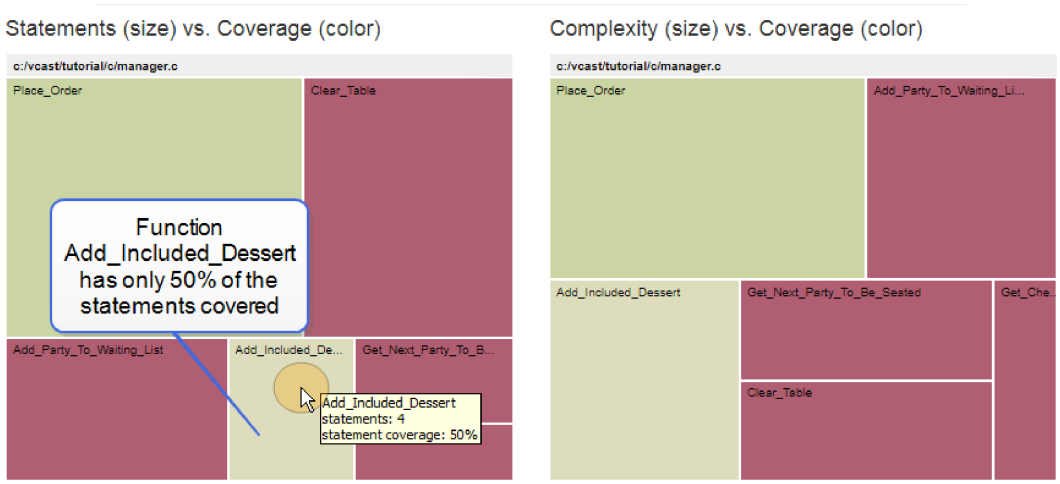
Metrics Display

Metrics are displayed in the right of the browser for the file or files selected in the Source Code Tree. In our example, we have selected the single file **`manager.c`**, and the dashboard displays the metrics associated with that file.

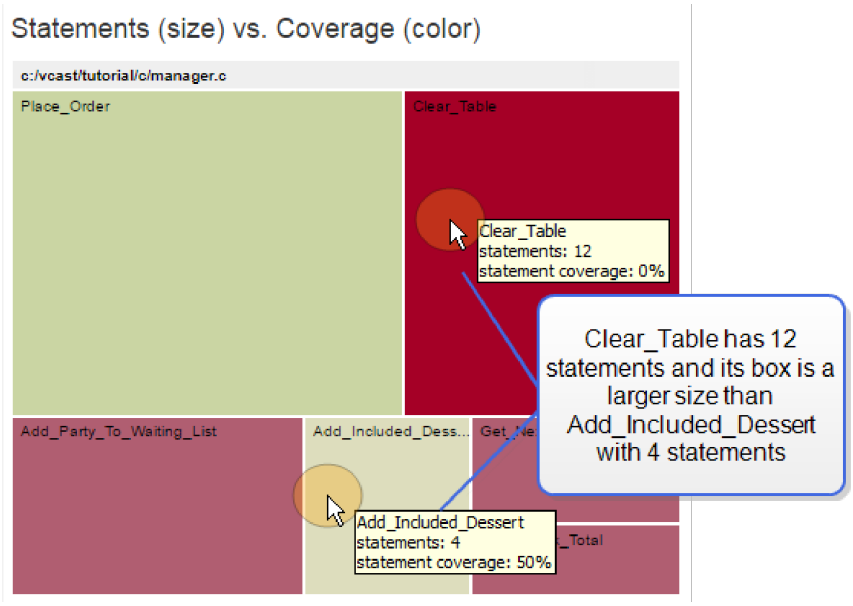
The top boxes provide the Metrics and Testing data for the selected file, **`manager.c`**. When Static Analysis data is available, a third box is provided.



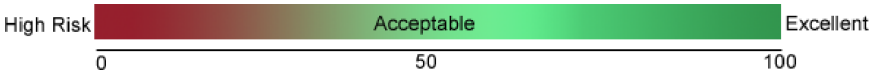
Below the boxes two treemaps are displayed for the file **`manager.c`**, one for Statements vs. Coverage and one for Complexity vs. Coverage. Each function maps to a box in the treemap. Hovering over a box displays the underlying data.



The size of the boxes within the treemaps reflects the number of statements or level of complexity of the functions. Functions with a large number of statements or high complexity will be larger in size.



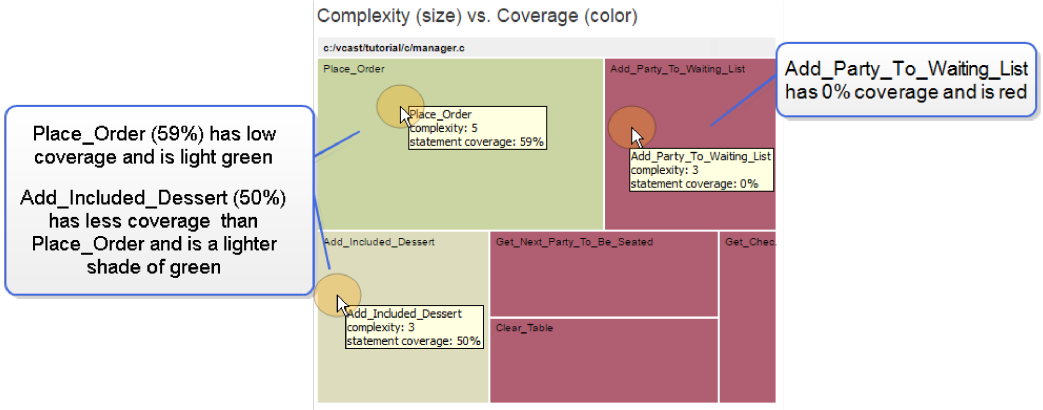
The color of the boxes within the treemap indicates the level of coverage of the functions.



> Green indicates a high level of coverage.

> Red indicates a low level of coverage.

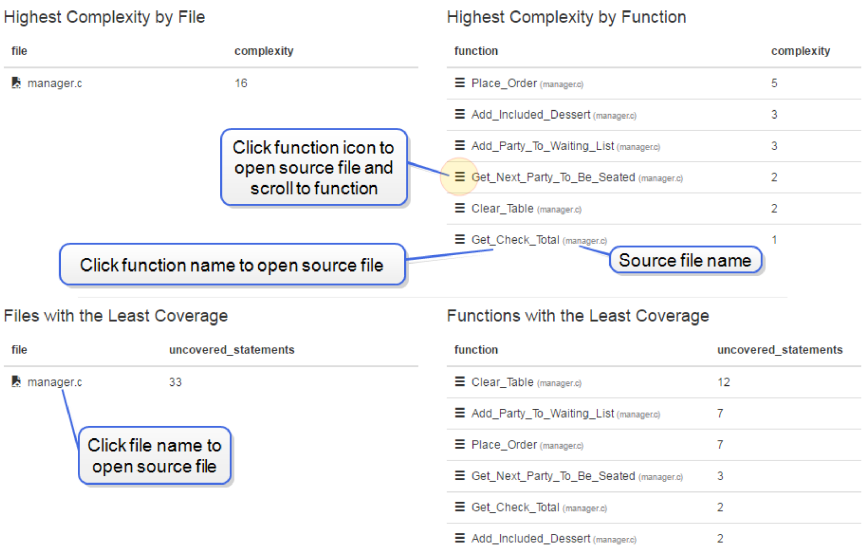
> Gradient shades reflect partial coverage, with less coverage and higher risk as the values approach red.



Using the example above, we can easily identify that the function Add_Party_To_Waiting_List is a hot spot which is highly complex and poorly tested. This information is critical in deciding how to best allocate testing and refactoring resources on a project.

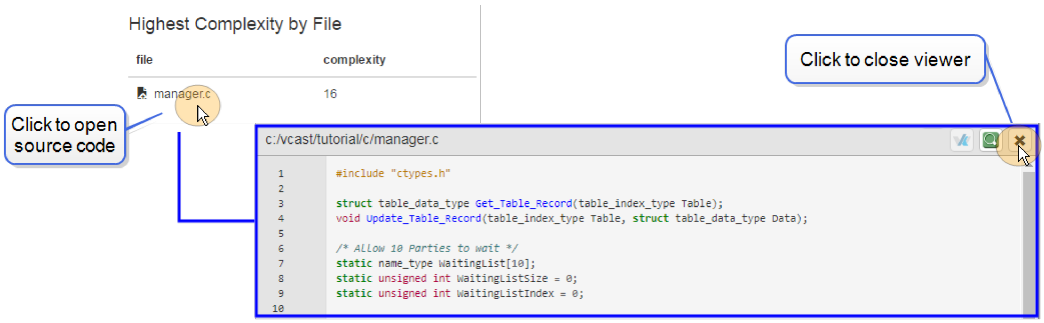
Tabular data is located by scrolling down the browser window. A set of four tables is provided showing the Highest Complexity listed by file and by function, and the Least Coverage listed by file and by

function for the selected files. In this example, we have selected to view the metrics for the manager.c unit. Note that the name of the source file is provided in parentheses to the right of the function name.



Source Code Viewer

Clicking on any of the listed source file names or the icon in either the Source Code Tree or the Metrics Tables will open the source code in a viewer. Clicking on the icon of a listed function will open the source file and scroll to the function. Use the button in the Title Bar to close the viewer.



Coverage Viewer

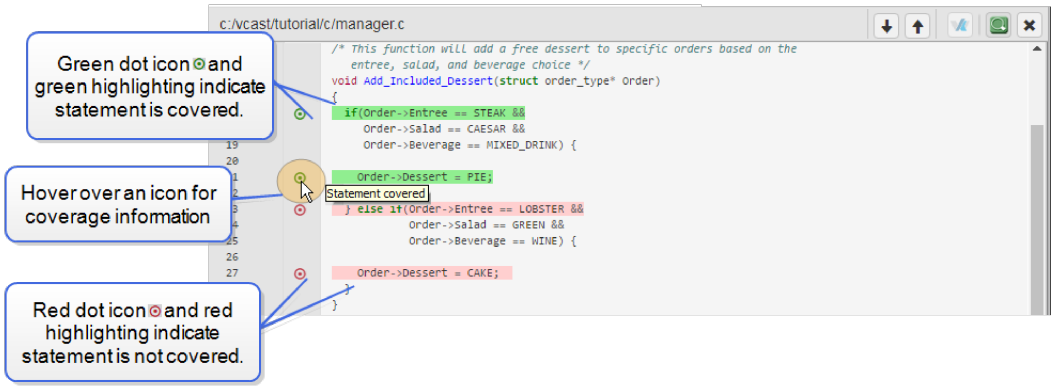
Select one of the following buttons from the upper right of the Source Code Viewer's Title Bar to view coverage for the source file:

- Opens the Coverage Viewer. This button is only available when the selected file has covered branches, pairs or statements.
- Opens Klocwork Analysis results. This button is only available when the selected file has Klockwork Analysis results.
- Closes the Viewer and returns to the Analytics dashboard.

- Jumps to the next uncovered or partially covered line.

- Jumps to the previous uncovered or partially covered line.

The Coverage Viewer provides an annotated version of the source file, colorized to indicate the coverage level achieved. Green highlighted code indicates the line is covered. Red highlighted code indicates the line is not covered. Yellow highlighted code indicates partial coverage for the line. In the example below, the file **`manager.c`** shows Statement coverage:



Icons in the column on the left give additional information regarding coverage. Hover over an icon for more information. The following icons (in combination with the red, green and yellow line highlighting) are used to annotate the coverage level:

For Statement coverage:

- Statement covered

- Statement not covered

For Branch coverage:

- True covered (No false branch)

- True and False covered

- True covered, False not covered

- True not covered, False covered

- Neither True nor False covered

For Merged coverage data:

- All covered

- Not covered

- Partial coverage

Close the Analytics Server

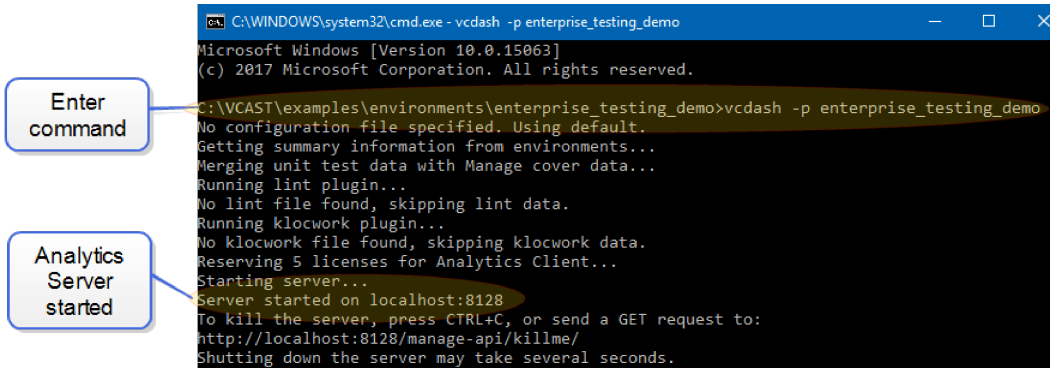
To close the Analytics Server, return to the DOS command prompt, and from the command line enter **Ctrl +C**.

Running the Analytics Server

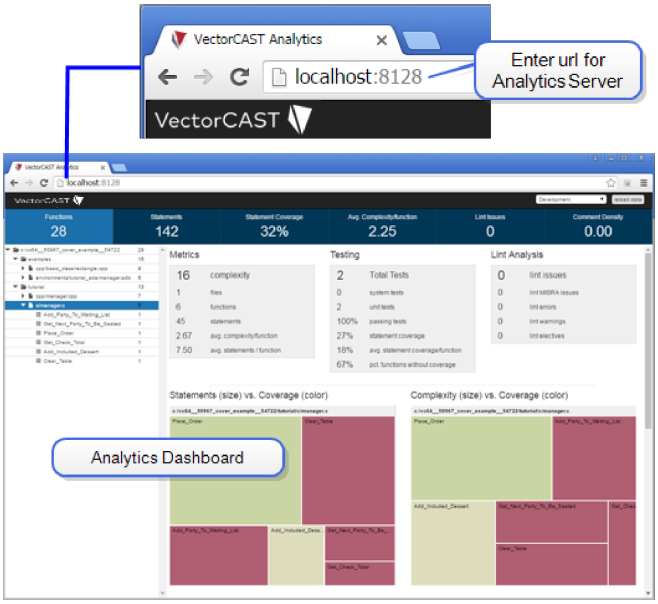
Running Analytics From the Command Line

To launch the Analytics Server, open a DOS command prompt by selecting the Command Prompt icon from the Toolbar. The **vcdash** command is run to launch the Analytics Server. The syntax for **vcdash** is as follows:

```
%VECTORCAST_DIR%/vcdash -p <project.vcm> [options]
```



Once **vcdash** launches the Analytics Server from the command line, open the Analytics Dashboard by first opening a web browser and then entering the web address for the Analytics Server. In our example the address is: **localhost:8128**. The dashboard opens in the web browser.



See "Analytics Server Options Reference" on page 22 for more information.

Tracking Trends and Project History

VectorCAST/Analytics has a history tracking system that keeps track of metric values over time. To

use the history tracking system, you must set up a history directory and add snapshots to that directory. Typically, a new history snapshot is saved at the end of every build/test cycle.

Create a History Directory

To create an initial history directory, add the first snapshot using the following command:

```
%VECTORCAST_DIR%/vcdash -p <project.vcm> --history-dir=<path-to-directory> --
save-history
```

Run this command after making changes to the project, running tests, or rebuilding to add subsequent snapshots.

Tip: Ideally, you should integrate Analytics with a build system such as Jenkins by running **`vcdash`** with the **`--save-history`** option at the end of every build/test cycle.

View History Trends on the Dashboard

When you have an existing history directory, invoke the Analytics Dashboard to view data trends. Use the following command:

```
%VECTORCAST_DIR%/vcdash -p <project.vcm> --history-dir=<path-to-directory>
```

Clicking the Reload button on the Dashboard reloads the newest snapshot.

The Dashboard reloads data when:

- A new history snapshot is saved
- A history snapshot is removed
- A history snapshot is edited.

Editing History Points

History points can be removed, renamed, or re-stamped with different times. An ID is required to modify a history snapshot. Use the following command to list all histories with their IDs:

```
%VECTORCAST_DIR%/vcdash --history-dir=<path-to-directory> --list-history
```

To name a history snapshot, use the following command. Snapshot names require a string without spaces.

```
%VECTORCAST_DIR%/vcdash --history-dir=<path-to-directory> --edit-history=<ID>
--name=<name-of-file>
```

To re-stamp a timestamp, use the following command. Timestamp format is UNIX epoch time in seconds (such as the **`date+%s`** UNIX command).

```
%VECTORCAST_DIR%/vcdash --history-dir=<path-to-directory> --edit-history=<ID>
--timestamp=1490207126
```

To remove a history snapshot, use the following command:

```
%VECTORCAST_DIR%/vcdash --history-dir=<path-to-directory> --remove-
history=<ID>
```

Including a Source Archive

By default, histories do not include a source archive. This is helpful if you do not want the source to be included, or if your source files are very large. This does, however, make the transfer of the history directory from one machine to another difficult, since the paths will not match the source installation on another machine.

To help with this issue, you can save a portable source archive with your snapshot. Note that only one archive exists in the history directory at a time. The source directory gets overwritten each time you save a new history.

Note: When using a source archive, coverage metrics work, but viewing covered/uncovered lines in the file view is not supported.

To archive the source files used in the Manage project along with the snapshot, use the following command:

```
%VECTORCAST_DIR%/vcdash --project=<project.vcm> --history-dir=<path-to-
directory> --save-history --include-source
```

Tip: A good practice is to save the source archive whenever you save a history snapshot.

Analytics Server Options Reference

The following options are available when running **`vcdash`**:

Option	Description
`s[--clients] arg <=5>`	The maximum number of clients allowed.
`-c [--config] arg`	The configuration file to use.
`[--coverdb] arg`	Cover database file **`<cover.db>`**. Can specify multiple

Option	Description
	cover databases.
[--create-config arg]	Create a new configuration in the specified directory.
-d [--dashboard-dir] arg	The dashboard directory.
[--edit-history arg	Edit history with the specified ID. Use with **--timestamp** or **--name**.
[--history-dir] arg	The history directory.
[--include-source]	When saving a history record, include the source files.
[--list-history]	Lists all history snapshots in the trend file and their IDs.
[--load-archive] arg	Loads the archive file.
[--mangle]	Mangle file and function names (not supported with history at this time).
[--name] arg	The history name. Used with **--edit-history** or **--save-history**.
-P [--port] arg (=0)	The port the server runs on.
-p [--project] arg	A Manage project **<.vcm>** Can support an aggregate display of data from multiple Manage projects. For example: **vcdash --project <project1> [[--project <project2>] ...]** The project argument can be specified by any of the following methods: **- <project>** **- <project>.vcm** All source files from the projects are displayed in the Dashboard, and data is merged (except coverage data).
[--remove-history] arg	Removes history with the specified ID.
[--run-server]	Starts the dashboard server. The default is on unless editing archives/histories. When using **--save-archive** or **--save-history**, the server does not start by default. This flag forces it to run.
[--save-archive] arg	Saves the archive file.
[--save-history]	Adds a loaded/saved archive into the directory specified by **--history-dir**.
[--single-license-fallback]	If there are not enough client licenses available, try again with one license.

Option	Description
[--source-archive]	Saves a source archive when saving a history snapshot. Replaces the existing archive in the history directory.
[--timestamp] arg	The history timestamp. Used with **--edit-history** or **--save-history**.
[--vcdb] arg	vcshell database file **<.vcdb>**. Can specify multiple databases. Note that when using a vcshell database you must apply the **addmetrics** vcutil command in order to get results in Analytics.

Configuring the Analytics Server

Creating a New Configuration

If you want to create several dashboards or customize server setup, the most convenient way is to create and modify a configuration directory. Once you create your configuration, you can then pass it as a **-c** argument when you invoke **vcdash** from the command line.

Create a new configuration using the **--create-config arg** option. This option creates a new custom configuration directory in the specified directory. The directory may be modified as needed for your configuration. For example:

```
%VECTORCAST_DIR%/vcdash --create-config=vcdash_config
```

The **vcdash** command takes a **-c** argument that passes in a JSON configuration file. The configuration file is a single object which contains global settings and three groups of child objects: Server, Filter, and Plugins. If you do not specify the configuration file, **vcdash** uses the default values provided in the tables below.

Global Settings

Setting	Type	Default	Description
dashboard	string/path	(install)	The directory to find dashboards. By default, this uses the vcdash installation directory.

Server Group Settings

Setting	Type	Default	Description
host	string	"0.0.0.0"	Valid host strings to contact server. "0.0.0.0" means accept any string or IP that maps to the server.
port	int	8128	Port on which to host the server. Make sure you have permission for the port (Linux) and open up any firewalls.

Filter Group Settings

The Filter Group is an array of filters, which are objects that explicitly include or exclude functions from the Dashboard. If any allowlists are defined, no files outside the allowlist are included. Analytics applies allowlists before denylists, but otherwise applies all filters in the order specified in the config file. Each filter has two fields.

Setting	Type	Default	Description
kind	string	required	"allowlist" or "denylist"
items	array	required	An array of regular expressions to be allowlisted or denylisted.

Regular expressions follow python regular expression syntax, and are matched against functions in the format **file_path/function_name**. For example, a function **foo()** in file **/home/users/sdf/main.c** is keyed as **/home/users/sdf/main.c/foo**.

To include only functions in **main.c**, you could create the following filter:

```
{
  "kind" : "allowlist",
  "items" : [
   "/home/users/sdf/main\.c/.*"
 ]
}
```

Plugin Group Settings

The Plugin Group may consist of any number of settings for plugins created by VectorCAST support personnel or created by customers. Due to their unique nature, these settings are not listed here. The setting is keyed off the plugin name, and the value is an object consisting of that plugin's settings.

For example, the following enables the **mydata** plugin and configures its **foo** setting to be 42:

```
"mydata" : {
   "foo" : 42
}
```

For more information on plugins, see "Adding Metrics With Plugins" on page 40.

Example Configuration File

In the following example configuration file, the host can be reached by **http://vectortools** on port 80 (assuming IT set up DNS to point to that location). **vcdash** will load the **mydata** plugin and pass the **foo** setting with the value 42, and load only data for functions in **/home/users/sdf/main.c**.

```
{
  "dashboard": "demo",
  "server": {
    "host": "vectortools",
    "port": 80,
    "session_timeout" : 5
},
  "plugins": {
    "mydata": {
      "foo": 42
    }
},
  "filters": [
    {
      "kind": "allowlist",
      "items": { "/home/users/sdf/main\.c/.*"]
    }
  ]
}
```

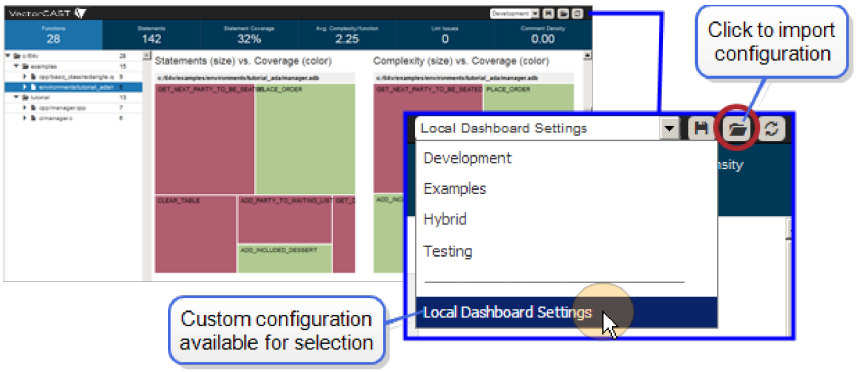
Customizing the Analytics Dashboard

Create A Custom Dashboard

VectorCAST/Analytics allows users to save a Dashboard, modify a Dashboard, or create a new Dashboard and then import the custom Dashboard.

To save a Dashboard, with a Dashboard open, choose the **Save Configuration** button in the upper right to export the current Dashboard to a JSON file. By default, it is named `local_dashboard.vdash`. You can then edit this Dashboard file to suit your needs. It is recommended you run it through a JSON validator.

When ready to import your custom Dashboard, select the **Load Configuration** button to import the `local_dashboard.vdash` file. It is then listed in the drop-down menu as "Local Dashboard Settings."



When using the **Save Configuration / Load Configuration** buttons on the dashboard, note that you cannot configure the server or set the configuration. If you want to make permanent changes, we recommend that you run the product from the command line.

For example, if you want to make several dashboards, create a configuration directory using the `--create-config arg` option (see "Creating a New Configuration" on page 26). This option creates a new custom configuration directory in the specified directory. You can then copy and modify any of the `.vdash` files inside that directory to create new dashboards.

Dashboard File Format

The Dashboard files define the layout of the page and are located in **%VECTORCAST_DIR%/python/vector/apps/Analytics/static/dashboards**.

To make your own custom dashboard file, copy the **demo.vdash** file located in the **/dashboards** directory and specify a full path to the new dashboard in the configuration's dashboard setting. Alternatively, you may keep the copy inside the **/dashboards** directory and specify the file name without the **.vdash** extension.

The tables below describe the settings for the Dashboard file.

Top-Level Settings

Setting	Type	Description
version	string	Version of the **vdash** file. Currently unused.
organization	metric	Organization field to use for grouping files under a hierarchy.
head_metrics	list of metrics	The metrics to use in the header and sort by in the hierarchy. You can have a maximum of 6 head metrics.
display_requirements	list of strings	The tags which must be satisfied for this dashboard to load or show in the drop-down.The following tags exist by default: • **has_lint** • **has_klocwork** • **has_statement_coverage** • **has_branch_coverage** • **has_requirements** You can set additional tags using plugins.
dashboard	object	See "Dashboard Settings" table below.

Dashboard Settings

The Dashboard setting is a complex object that defines the overall dashboard. Its top-level settings are:

Setting	Type	Description
name	string	Currently unused.
description	string	Currently unused.
rows	list of objects	See "Rows Settings" table below.

Rows Settings

The Rows setting is a list of complex objects that define the rows in the Dashboard. Rows are

displayed in the order in which they are defined, from top to bottom. Its top-level settings are:

Setting	Type	Description
name	string	Currently unused.
widgets	list of objects	See "Widgets Settings" table below.

Widgets Settings

The Widgets setting defines the list of widgets which are displayed on the Dashboard. Settings vary depending on the display setting. Widgets are displayed in the order in which they are defined, from left to right. Widget settings are:

Setting	Type	Description
name	string	Title over the widget.
description	string	Mouseover tooltip for the title.

widget_type	string	The type of widget to display and the fields it supports (other than name and description).
		
		• **summary**
		  ◦ **grouping**
		  ◦ **metrics (array)**
		• **treemap**
		  ◦ **grouping**
		  ◦ **metrics (array of length 2)**
		  ◦ **colors**
		  ◦ **color_scale**
		• **tablegroup**
		  ◦ **grouping**
		  ◦ **metrics (array)**
		  ◦ **size**
		  ◦ **hide zeroes**
		• **piechart**
		  ◦ **grouping**
		  ◦ **metrics (string, not an array)**
		• **barchart**
		  ◦ **grouping**
		  ◦ **metrics (string, not an array)**

grouping	metric	The metric that defines how to group the data. For most currently used widgets, this is "func", meaning group data by each individual function.
metrics	metric or list of metrics	For most widgets, this is a string that

Setting	Type	Description
		defines the metric we want to reduce the group down by. Some widgets, such as treemaps and summaries, display 2 or more reductions; in which case, the setting is defined as a list of metrics. For example: • In a table, a **grouping** of `func` and a **metrics** of `tests_failed` groups by function, and provides their summed **tests_failed**. • In a table, a **grouping** of `group_file` and a **metrics** of `tests_failed` groups by file, and provides their summed **tests_failed**. • In a bar chart, a **grouping** of `group_complexity` and a **metrics** of `tests_failed` groups by how complex the function is, then displays a bar chart of each group's summed **tests_failed**.
`color_scale`	list of ints or `relative`	For widgets that are showing relative differences, such as a treemap, the system needs to know what the lowest and highest values are. This setting defines them in an array as `[lowest,highest]`. Using `relative` sets the lowest and highest value to the lowest and highest in the set. Treemaps are not currently supported.
`colors`	string or list of strings	Specifies the color template for the widget in the format `template.size,` where `template` is a named template, and `size` is the number of color steps. For example, `Reds.8`. The system will scale your input across the steps of color. See "Color Templates" on page 36 for details. Alternatively, you can provide a list of strings that are colors in hex format: ["#FF0000", "#00FF00", "#0000ff"].
`hide_zeroes`	boolean	Ignore groups with the value 0.

Setting	Type	Description
`display_requirements`	list of strings	The conditions which must be satisfied for this widget to display. See Dashboard display_requirements above.

Supported Groups

The following groups are available by default. To define new groups, you must write a plugin. See "Adding Metrics With Plugins" on page 40 for more information.

Group	Description
`function`	Default. Groups data by each function.
`group_file`	Groups data by each file.
`group_complexity`	Groups data by each function's complexity (<5, <11, <21, <51, >50)
`group_coverage`	Groups data by each function's level of coverage (0%, <25%, <50%, <75%, <100%, 100%)

Supported Metrics

Supported metrics are provided in the following tables. The "Granularity" column identifies the individual records which make up the metric (file, function, or requirement).

When selecting functions in the dashboard, widgets using metrics with file or requirement granularity display that object's value. When a selection encompasses a number of functions, the data is aggregated over the unique set of files or requirements those functions are related to.

Standard Metrics

Metric	Granularity	Description
`avg_complexity`	function	Average complexity per function
`avg_covered_pct`	function	Average statement coverage per function
`blank_lines`	file	Number of blank lines (requires `vcdb` after using the `addmetrics` command)
`branches`	function	Number of branches
`code_lines`	file	Number of code lines (requires `vcdb` after using the `addmetrics` command)
`comment_lines`	file	Number of comment lines(requires `vcdb` after using the `addmetrics`

Metric	Granularity	Description
		command)
comment_source_ratio	file	Comment lines / code lines (requires **vcdb** after using the **addmetrics** command)
complexity	function	Complexity
control_flow_total	function	Total control flows
count	function	Number of functions
coverable_functions	function	Number of coverable functions
covered_branches	function	Number of fully-covered branches
covered_function_calls	function	Number of covered function calls
covered_functions	function	Number of covered functions
covered_pct	function	Statement coverage
covered_statements	function	Number of statements covered
expected_total	function	Total expected values
failed_control_flow	function	Failed control flows
failed_expected	function	Failed expected values
file_count	file	Number of files
file_without_test_count	file	Number of files without tests
function_call_coverage	function	Percentage of covered function calls
function_calls	function	Number of function calls
function_coverage	function	Percent of covered functions
functions_without_coverage	function	Number of functions without coverage
mcdc_branches	function	Number of MC/DC branches (conditions)
mcdc_covered_branches	function	Number of fully-covered MC/DC branches
mcdc_covered_pairs	function	Number of fully-covered MC/DC pairs
mcdc_pairs	function	Number of MC/DC pairs
partial_branches	function	Number of branches with both true/false coverable, but only one or the other is covered
pct_covered_branches	function	Percentage of fully-covered branches
pct_functions_without_coverage	function	Percentage of functions without coverage
pct_functions_without_tests	function	Percentage of functions without tests

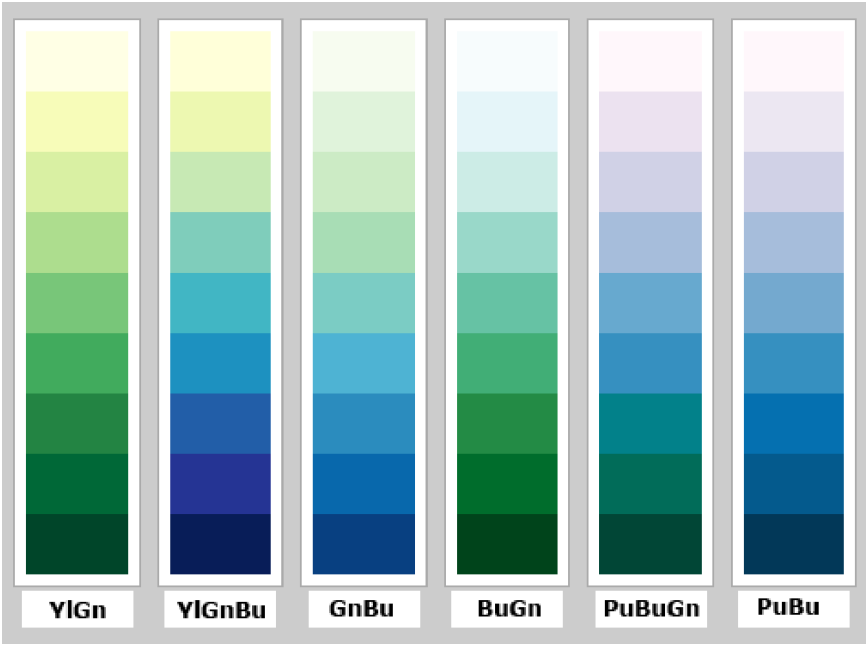
Metric	Granularity	Description
pct_mcdc_covered_branches	function	Percentage of fully-covered MC/DC branches
pct_mcdc_covered_pairs	function	Percentage of fully-covered MC/DC pairs
pct_remaining_tests	function	Testing completeness
pct_requirements_passed	requirement	Requirements_passed / requirements
pct_requirements_tested	requirement	Percentage of tested requirements vs. number of requirements in the repository
remaining_tests	function	Tests needed
requirements	requirement	Number of requirements tested
requirements_passed	requirement	Number of requirements tested where all tests pass
signals	function	Total signals
statements	function	Number of statements
tests_failed	function	Tests failed
tests_passed	function	Tests passed
tests_passed_pct	function	Percentage of passed tests
tests_skipped	function	Tests skipped
tests_total	function	Number of tests
tests_with_expected	function	Tests with expected values
tests_with_expected_and_reqs	function	Tests with expected values and requirements
tests_with_expected_no_reqs	function	Tests with expected values but no requirements
total_lines	file	Number of lines (requires **vcdb** after using the **addmetrics** command)
uncovered_function_call_pct	function	Percentage of uncovered function calls
uncovered_function_calls	function	Number of uncovered function calls
uncovered_function_pct	function	Percentage of uncovered functions
uncovered_functions	function	Number of uncovered functions
uncovered_pct	function	Percentage of statements not covered
uncovered_statements	number	Number of statements not covered
unit_tests	function	Number of unit tests

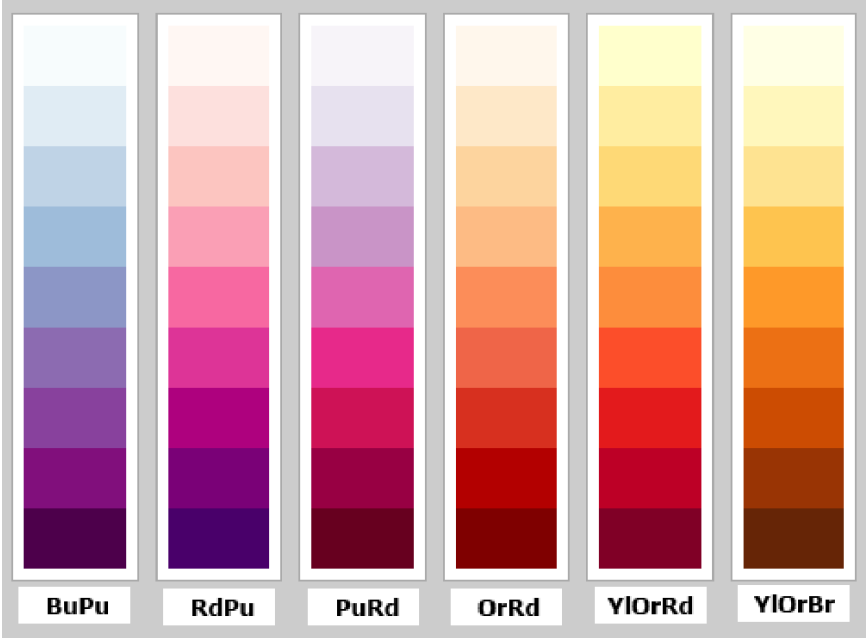
Static Analysis Plugin Metrics

Metric	Granularity	Description
`klocwork_avg_issues_per_function`	function	Average number of Klocwork issues per function
`klocwork_clean_functions`	function	Total number of functions with zero Klocwork issues
`klocwork_errors`	function	Klocwork errors
`klocwork_info`	function	Klocwork info
`klocwork_misra`	function	Klocwork MISRA issues
`klocwork_pct_clean_functions`	function	Percentage of functions with zero Klocwork issues
`klocwork_total_issues`	function	Klocwork total issues
`klocwork_warnings`	function	Klocwork warnings
`lint_avg_issues_per_function`	function	Average number of Lint issues per function
`lint_clean_functions`	function	Total number of functions with zero Lint issues
`lint_errors`	function	Lint Errors
`lint_info`	function	Lint Electives
`lint_misra`	function	Lint MISRA Issues
`lint_pct_clean_functions`	function	Percentage of functions with zero Lint issues
`lint_total_issues`	function	Lint Issues
`lint_warnings`	function	Lint Warnings

Color Templates

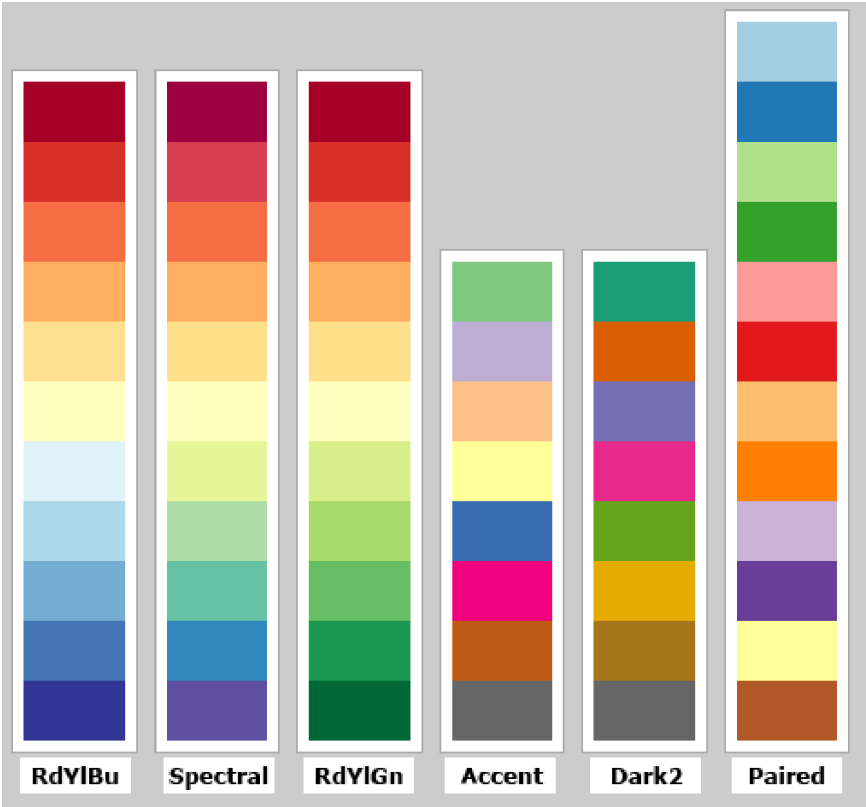
The color template is specified in the Dashboard file. See Widget Settings > "colors" on page 32 for more information.

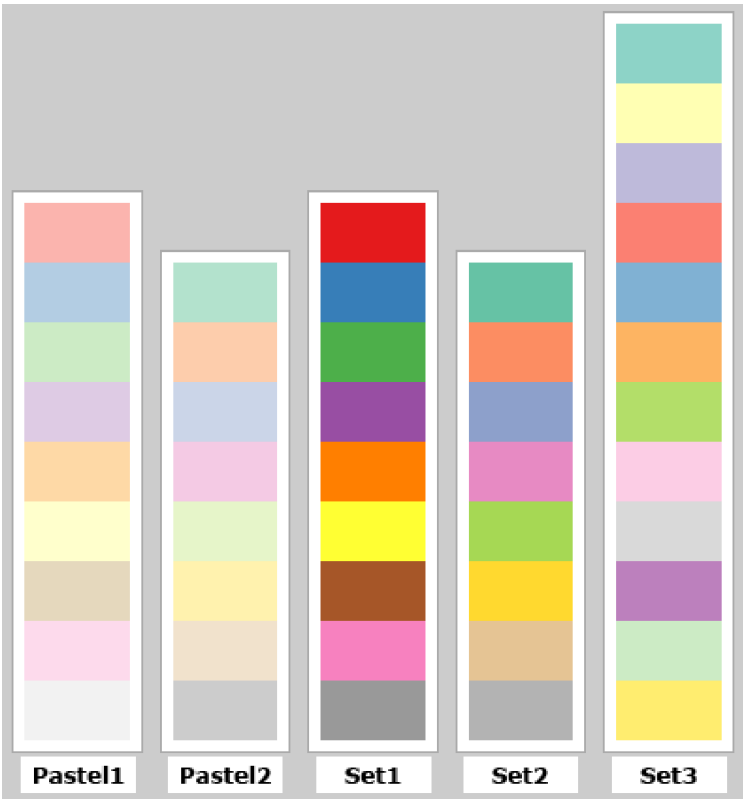

YlGn
YlGnBu
GnBu
BuGn
PuBuGn
PuBu


BuPu
RdPu
PuRd
OrRd
YlOrRd
YlOrBr


Purples
Blues
Greens
Oranges
Reds
Greys


PuOr
BrBG
PRGn
PiYG
RdBu
RdGy


RdYlBu
Spectral
RdYlGn
Accent
Dark2
Paired


Pastel1
Pastel2
Set1
Set2
Set3

Adding Metrics With Plugins

Analytics Plugin System

VectorCAST/Analytics has a Plugin system which allows you to attach data to files or functions and define new metrics and groups. Plugins allow you to:

- Attach data to files and functions
- Define new metrics
- Define new groups
- Add tags to control what is and is not displayed on a Dashboard

Note: Contact the VectorCAST Technical Support team for additional support in creating your specific plugins:
Email: support@vector.com
Web: www.vector.com/support

Index