GNATdashboard Documentation

Release 27.0w

AdaCore

Apr 28, 2026

CONTENTS

CHAPTER

ONE

GNATDASHBOARD QUICK COOKBOOK

1.1 Getting Started with GNATdashboard

1.1.1 Prerequisites

- a **SonarQube Scanner** installation configured for the targeted **SonarQube** instance, and in your `$PATH` (see *Prerequisites* for more information on supported versions of **SonarQube** and **SonarQube Scanner**).
- a **GNAT Pro** installation (the most recent installed on the system), and in your `$PATH`
- *(optional, and recommended)* a **GNAT SAS** installation, in your `$PATH`

1.1.2 GNATdashboard setup

- *(on Windows)* run the installer, and place `<install_prefix>bin` in your `$PATH`
- *(on *NIX)* extract the installation directory from the archive, and place `<install_prefix>/bin` in your `$PATH`
- *(on all platforms)* copy `<install_prefix>/share/sonar/extensions/plugins/*.jar` into the equivalent place in your **SonarQube** installation, and restart **SonarQube**

1.1.3 The 5-lines manual to GNATdashboard

GNATdashboard contains a driver program, **GNAThub**, which:

- executes all **GNAT** tools and stores the results in a database
- creates a configuration file for your project ready to use by **sonar-scanner** (`sonar-project.properties`)
- launches the **sonar-scanner**

The **SonarQube Scanner** is reading the results from the database created by the driver **GNAThub**.

For more information, the full manual is available at:

- `<install_prefix>/share/doc/gnatdashboard/html`
- `<install_prefix>/share/doc/gnatdashboard/pdf`

And online at https://docs.adacore.com/gnatdashboard-docs/.

1.1.4 Trying with an example

A complete example is provided at:

- `<install_prefix>/share/example/gnatdashboard/sdc`

This contains an Ada project and a `Makefile` which builds the project, computes coverage information using **Gcov**, then launches the **GNAThub** driver.

The `Makefile` also launches a Python script which does a simple textual dump of the contents of the database - this is to demonstrate the scripting capabilities of the **GNATdashboard** driver.

CHAPTER

TWO

GNATDASHBOARD ADMINISTRATOR AND USER'S MANUAL

2.1 Introduction

GNATdashboard is a product for monitoring the software quality of Ada projects.

GNATdashboard fits naturally into a software development team's workflow by leveraging on project files to configure, run, and analyze the output from **GNAT** tools. Its driver program processes data such as compiler warnings, **GNAT SAS** diagnostic messages, style check violations, and coverage data, and makes it available for reference and analysis through its local database.

GNATdashboard works as follows:

- it runs tools from GNAT Pro tool suite such as **GNAT Check**, **GNAT Metric**, **GNAT Coverage** and additional tools like **GNAT SAS**
- it processes the output from those tools and collects it in a database
- it extracts data from the database and feeds it to high-level platforms such as **SonarQube** and **Squoring**

GNATdashboard is available on the following platforms:

- Windows (32-bits and 64-bits)
- Linux (32-bits and 64-bits)
- OS X 10.10 (64-bits)

2.1.1 Prerequisites

To use **GNATdashboard** you must have both the **GNAT** tools you intend to run on your project (such as **GNAT Metric** and **GNAT Check**) and **GNAThub** installed.

In order to visualize results in the **SonarQube** platform, the corresponding plug-in must be deployed.

Sonar Ada Plugin is supported for versions of **SonarQube** and **SonarQube Scanner** supported by **SonarSource**:

- for **SonarQube** 2025.1 LTA
- **SonarQube Scanner** 7.0.2.4839

2.1.2 Upgrading from GNATdashboard 1.0

Various breaking changes to the **Sonar Ada Plugin** were mandatory to adapt to the new **SonarQube** plug-in API and support the latest versions of **SonarQube** (from LTS to stable). Is this regard, the **Sonar Ada Plugin** is no longer compatible nor supported on versions of **SonarQube** prior to 2025.1.

Note that **GNATdashboard** 1.1.x is required for use with the latest version of the **Sonar Ada Plugin**.

2.1.3 Release

The current version is 27.0w.

2.1.4 Installation

Download the **GNATdashboard** package using your **GNAT Tracker** account.

On Windows, run the graphical installer. On other platforms, un-zip the downloaded archive and install it on your system (usually at some location such as `/usr/local/gnatpro`) under a new folder. In order to be able to use it, you should add to your `$PATH` the `/bin/` folder of your **GNATdashboard** installation.

The **SonarQube** plug-in is located in the directory `<install_prefix>/share/sonar`. This needs to be installed manually. In order to do that, you should

- stop the **SonarQube** server
- copy the Sonar Ada plugin from **GNATdashboard** installation repository subfolder `/share/sonar/extensions/plugins/` into the **SonarQube** installation repository under `/extensions/plugins/`
- restart the **SonarQube** server.

Note: It is very important that only one **|Sonar Ada plugin|** be present in this repository and it needs to match the **GNATdashboard** version that you just installed. Any older version of this plugin must be removed before restarting **SonarQube** server.

2.1.5 Setting your environment

To use **GNAThub** you must have all the tools you are planning to use on your project in your `$PATH` (some tools come with the **GNAT** distribution, while others are available as separate packages).

2.2 How to execute the GNAThub driver

2.2.1 Getting started

Execute **GNAThub** as you would other **GNAT** tool. In most cases, you will provide a **GNAT** project file (`.gpr`):

```
$ gnathub -P my_project.gpr
```

This executes each **GNAThub** plug-in for the project, collects the results of each tool, and stores those results in its local, temporary, **SQLite** database.

SQLite is a software library implementing a self-contained, serverless, zero-configuration, transactional SQL database engine. This makes it a perfect fit for storing and organizing the results of a single analysis and making those results available to a wide range of code quality management platforms.

Note: It is best to consider the **SQLite** database a blackbox.

This local, temporary, database is intented to be manipulated directly by **GNAThub** to store its data, and the recommended way to query data is by using the Python API exposed by the tool.

GNAThub will attempt to **reset** the entire database each time it is executed, unless given the switch **--incremental** that is designed to aggregate the results of different tools into the same database for cases where multiple passes cannot be avoided, *eg.*:

```
$ gnathub -P project --plugins gnatmetric
$ gnathub -P project --plugins gnatcheck,gnatcoverage --incremental
$ gnathub -P project --plugins sonar-config,sonar-scanner --incremental
```

Make sure to use the same project across all invocations of gnathub with the **:commmand:`--incremental`** switch.

It is important to note that **GNAThub** assumes the following properties on the **SQLite** database:

- It must contain at most one set of results per tool;
- It must contain results for one and only one project and set of sources;
- It must not be used for persistence (dedicated application such as **SonarQube** are intended for this use);
- Its schema can change from a version to another.

2.2.2 Outputs

There are two kinds of log file:

1. tools output
2. GNAThub execution log

(1) Files located in `<object_dir>/gnathub/logs` are output of tools, *e.g.* `codepeer.log` contains the output of the latest **GNAT SAS** run (provided that it was invoked from **GNAThub**). These files are generated if you invoke the tool using the API function *GNAThub.Run()*.

(2) GNAThub uses the same log mechanism as **GNAT Studio**: the output behavior can be customized through a configuration file. See the GNATcoll Traces documentation for more information.

2.2.3 Project file attributes

The **GNAThub** driver expects a number of attributes, some required and some optional, to be set in the project file.

General attributes

Plugins

List of the names of plug-ins to load and execute for this project. **GNAThub** ignores this attribute if you specify the **--plugins** switch on the command line. You can tailor this list by specifying the **Plugins_Off** attributes.

Plugins_Off

List of plug-ins names to remove from the execution queue if present. Use this to disable one or more plug-ins in the context of a specific project. This filter is applied after the computation of the complete plug-ins list.

Local_Repository

Path to a directory containing custom plug-ins to add to the execution queue of **GNAThub**. This is a supplementary list of plug-ins, which means it will extend the initial plug-ins list, computed from the **[system]** and **[global]** repositories.

The complete list of repositories can be found in *GNAThub.repositories()*.

SonarQube-specific attributes

Project_Name

Name of the project. This is used by the `sonar-config` plug-in to override the default project name provided by the **GNAT** project file.

Project_Version

Version of the project. This is used by the `sonar-config` plug-in to override the default project version set by the plug-in.

Project_Key

Unique key of the project. This is used by the `sonar-config` plug-in to override the default project key generated by the plug-in (from the project's name). This is a required attribute that allows **SonarQube** to discriminate between projects in its database.

Source_Encoding

Encoding to use to read files from this project. This is used by the `sonar-config` plug-in to override the default encoding set (**UTF-8**) and forward this value to **SonarQube** (which takes care of reading and indexing all source files).

Example

An example is worth a thousand words:

```
project My_Project is

   [...]

   package Dashboard is
      for Project_Name use "My_Custom_Project_Name";
      --  Translate to SonarQube Scanner property: sonar.projectName

      for Project_Key use "Custom_Key_For_SonarQube";
      --  Translate to SonarQube Scanner property: sonar.projectKey

      for Project_Version use "1.9.0";
      --  Translate to SonarQube Scanner property: sonar.projectVersion

      for Source_Encoding use "UTF-8";
      --  Translate to SonarQube Scanner property: sonar.sourceEncoding

      for Plugins_Off use ("GNATSAS");
      --  Disable GNATSAS plug-in

      for Local_Repository use "extra/gnatdashboard_plugins";
      --  Declare a local repository where GNAThub will look for additional
      --  plug-ins.
   end Dashboard;

end project
```

2.2.4 GNAThub's core plug-ins

GNATdashboard's driver comes with a set of core plug-ins, available in the **[system]** repository, allowing quick integration with a software development team's workflow.

The following tools are currently supported by the **GNAThub**'s core plugins:

2.2.5 GNAThub's additional plug-ins

An additional **[global]** repository is available for the user to store plugins. **GNAThub** searches this directory looking for additional plugins to load. This directory is never overwritten by an update, making it a good place to store custom plug-ins.

2.2.6 GNAThub's command line

You can specify switches on the command line to tune each execution of the driver. Please use **gnathub --help** to see the full list of supported switches.

--plugins

Expects a comma-separated list of plug-in names as argument. This list is used as the initial execution queue of the driver, replacing the list of plug-ins computed from the project attribute **Plugins**. However, the project attribute **Plugins_Off** still applies to that list and removes any explicitly disabled plug-ins.

--exec

Expects a Python file as argument. Executes the specified Python file after that the database has been updated. This allows for post-processing, possibly on the local database, using the exposed Python API. This implies implicit **--incremental** (retains any previous database) since a post-processing is performed on an already existing database and the incremental behavior will apply automatically when the **--exec** switch is used.

--incremental (short option -i)

Takes no argument. Instead of its default behavior, when you specify **--incremental**, **GNAThub** retains any previous database (instead of clearing it). The **GNAThub** database is intended to represent the results of analysis tools on one single project, so you need pass the same projects to all invocations of *gnathub* with the **--incremental** switch.

--dry-run (short option -n)

Takes no argument. Instead of its default behavior, when you specify **--dry-run**, **GNAThub** will report what changes it would have made and what plugins it would have executed rather than making and executing them. Check mode is a simulation that can be useful for testing the validity of a command line and project configuration without running the actual tools.

--gnatcheck-hide-exempted

Takes no argument. This is a GNATcheck specific switch, to be used when GNATcheck plugin is executed, otherwise will have no effect. This switch allows you to specify whether GNATcheck exempted wiolations should be taken into account and added into **GNAThub** database. Instead of the default behavior (where all exempted violations are imported), when you specify **--gnatcheck-hide-exempted**, **GNAThub** will ignore all GNATcheck violation reported in the "Exempted Coding Standard Violations" section of GNATcheck output file.

--target

Expects the target name as argument. Used to specify an alternate toolchain.

--subdirs

Expects a directory as argument. The database will be located in this directory relatively from the project object directory. Creates the subdirectory if necessary.

--collect-project-sources

Takes no argument. Indicates to the driver that the project sources should be collected and will create a copy of the current project in a folder named *sonar* in the **GNAThub** execution folder located under the project object directory. This is the folder where the data needed by the `sonar-scanner` will be collected and used by the **Sonar Ada Plugin** to upload analysis results into **SonarQube** dashboard. This switch usage might be of interest when not all the project sources are available under the current directory. However, this means that the sources as seen by SonarQube will not be in the original location in the project. If the switch is not explicitly passed in the *gnathub* command line, no local copy is created for the sources and the original files will be used by the **Sonar Ada Plugin**.

--sonar-work-dir

Expects a folder path as argument. This is the directory where the data needed by the `sonar-scanner` will be collected and used by the **Sonar Ada Plugin** to upload analysis results to **SonarQube** dashboard. Creates the directory if necessary.

--RTS

Expects an Ada runtime as argument. Forces a specific runtime.

--jobs (short option -j)

Expects the maximum number of processes to be executed concurrently as argument. Similar to the **-j** switch passed to **make**. **0** is a special value meaning "as many processes as possible". The default is **1**.

--targs:

Expects a command line program name and a list of switches to pass to that program, eg.:

```
$ gnathub --targs:gnatsas --verbose -- -P project.gpr
```

All switches following **--targs:<tool>** are passed to *<tool>*, stopping either at the sentinel – or at another **--targs:** option.

`--runners-only`

Takes no argument. Instead of its default behavior, when you specify **`--runners-only`**, **GNAThub** will only execute plugins that implement the *GNAThub.Runner* interface. This interface should be implemented by plugins that need to execute tools to produce results that will be then analysed by a plugin implementing the *GNAThub.Reporter* interface. Note that the same plugin can implement both interfaces to encapsulate all the logic to run a tool and collect its results in the same class.

`--reporters-only`

Takes no argument. Instead of its default behavior, when you specify **`--reporters-only`**, **GNAThub** will only execute plugins that implement the *GNAThub.Reporter* interface. This interface should be implemented by plugins that need to process the output of the tool to save its results within **GNAThub**. Note that the same plugin can implement both interfaces to encapsulate all the logic to run a tool and collect its results in the same class.

`-d`

Display progress for use in IDEs. This command is mainly intended for integration in IDEs, e.g. GNAT Studio.

2.3 Writing a GNAThub plug-in

2.3.1 Location for user-defined plug-ins

Store your plug-ins in the `extra` directory. **GNAThub** attempts to load all files in this directory except the ones whose name starts with an underscore (_), which are expected to be support files referenced by multiple plug-ins.

2.3.2 Structure

A **GNAThub** plug-in is a Python class that extends the *GNAThub.Plugin* abstract class. It must override the `GNAThub.Plugin.execute()` method and set the **`name`** property.

Additionally, the user can override the two following methods:

- *GNAThub.Plugin.setup()*
- *GNAThub.Plugin.teardown()*

These will be called respectively before and after the `GNAThub.Plugin.execute()` method.

2.3.3 Execution

The plug-in is discovered and loaded by the **GNAThub** driver unless explicitly disabled in the project file using the **`Plugins_Off`** attribute. If it remains enabled, it is executed along with the other plugins without any further action.

2.3.4 Logging

Plug-ins can integrate with the logging mechanism provided by the **GNAThub** API through the `log` property of the *GNAThub.Plugin* class, *e.g.*:

```
self.log.debug('resource found at %s', resource)
```

Note that the **GNAThub** API provides its own logging.Handler implementation to integrate with the standard Python logging facility meaning that one can use the `logging` Python module directly and automatically benefit from this integration. This becomes particularly useful when importing thirdparty modules that already rely on this logging facility (*e.g.* Python requests).

2.4 Integrating GNATdashboard into your workflow

GNATdashboard provides an integration with the **SonarSource** software. **SonarQube** is an open platform to manage code quality. It is extensible with plug-ins to add support for new languages and rules.

The **Sonar Ada Plugin** provided with **GNATdashboard** packages supports **SonarQube** platform for the LTA (long-term support) version (currently 2025.1).

2.4.1 How GNAThub integrates with SonarQube

Sonar Ada Plugin

SonarQube Scanner requires a **SonarQube** Ada plug-in to work on Ada sources. **Sonar Ada Plugin** is provided to this effect and is part of the **GNATdashboard** package. It is configured to read the **SQLite** database populated by **GNAThub** and its plug-ins (it reads its configuration in `sonar-project.properties` generated by **GNAThub** **sonar-config** plug-in).

To use it in the context of **GNATdashboard**, first deploy **Sonar Ada Plugin** into your **SonarQube** instance (please refer to **SonarQube** manual for plug-ins installation).

The **SonarQube** dashboard integrates technical debt information, which can be used to assess the amount of effort needed to fix the reported problems. For the Ada language, these values are reported through the **Sonar Ada Plugin**. This information is available to any analysis tools that might use these values as part of their analysis. Note that, for most rules violation, the remediation cost is constant per issue, and it depends on the rule severity (one one of the following values: Trivial, Easy, Medium, Major, High or Complex).

SonarQube Scanner

GNAThub execution (see *Getting started*) generates one **SonarQube Scanner** configuration file `sonar-project.properties` describing the project being analyzed (project name, project key, location of the `gnathub.db` file, ...). This is fully generated by the **sonar-config** plug-in of **GNAThub** and is expected to be used *as-is* by the **Sonar Ada Plugin** part of the **GNATdashboard** product.

GNAThub **sonar-config** plug-in regenerates the `sonar-project.properties` configuration file each time it is executed. This file is a project configuration file and does not contain authentication credentials in it. In order to authenticate, these parameters could be passed explicitly when the **sonar-scanner** plug-in is executed through **--targs:** switch tool specific parameters, *eg.*:

```
$ gnathub -P project --plugins sonar-scanner --incremental
       --targs:sonar-scanner -Dsonar.projectKey=<project_key>
       -Dsonar.login=<sonar_login> -Dsonar.password=<sonar-password>
```

Note: For **SonarQube** 2025.1 LTA version, in order to be able to upload analysis results into the dashboard the authentication information is mandatory and should be provided when the **sonar-scanner** plug-in is executed as **--targs:** switch parameters. Instead of passing through explicit login/password information, we recommend the usage of tokens to authenticate. Once generated, the token could be passed as **--targs:** parameter, *eg.*:

```
$ gnathub -P project --plugins sonar-scanner --incremental
       --targs:sonar-scanner -Dsonar.projectKey=<project_key>
       -Dsonar.login=<myAuthenticationToken>
```

To know more about how to generate and use the tokens please refer to the official **SonarQube** documentation related to *Generating and Using Tokens* at https://docs.sonarqube.org/latest/user-guide/user-token/.

SonarQube Scanner reads its settings from two different files:

- the system-wide `$SONAR_RUNNER_HOME/conf/sonar-scanner.properties`
- the project-specific `sonar-project.properties` file provided through the `-Dproject.settings` command line argument of **SonarQube Scanner**

See Analyzing with SonarQube Scanner for more information on **SonarQube Scanner**.

By default, **GNAThub** will create the **SonarQube Scanner** configuration file, and launch the **SonarQube Scanner** itself.

This execution is handled by the **sonar-scanner** plug-in of **GNAThub**. This is always scheduled as the last **GNAThub** plugin: this ensures that the output of all other tools is available in the **GNAThub** database and that `sonar-project.properties` has been generated prior to launching the **SonarQube Scanner**.

Note: **GNAThub** expects **sonar-scanner** or **sonar-scanner.bat** to be available on the `$PATH`.

2.4.2 GNATdashboard without SonarQube

SonarQube is only used to display results from various analyses.

The technical debt information is also available in the **SonarQube** dashboard for the issues reported by the supported tools. It is therefore not mandatory to display the analysis: for instance, you can choose not to use the **SonarQube** integration but use custom scripting to extract the analysis results stored in the **GNAThub** database.

To disable the **sonar-config** and **sonar-scanner** plug-ins, use the **Plugins_Off** project attribute (see *Plugins_Off*).

2.4.3 Incremental analysis

GNAThub can be configured to run one plug-in at a time and thus provides incremental execution of each plug-in (allowing for finer grain control and better integration in existing project and code base).

Use *--incremental (short option -i)* to enable incremental mode:

```
$ for plugin in gnatcheck gnatmetric codepeer; do
>   gnathub --incremental --plugins $plugin -P project.gpr
> done

$ gnathub --incremental --plugins sonar-config -P project.gpr
```

(continues on next page)

(continued from previous page)

```
$ gnathub --incremental --plugins sonar-scanner -P project.gpr
        --targs:sonar-scanner -Dsonar.login=<myAuthenticationToken>
```

If you do not wish to use the **SonarQube** integration, you can simply omit the last two **GNAThub** executions and use the *--exec* switch for custom database processing:

```
$ gnathub --exec my-results-collector.py -P project.gpr
```

CHAPTER

THREE

GNATDASHBOARD SONAR ADA PLUGIN

3.1 Sonar Ada Plugin

This SonarQube plugin loads the analysis of Ada projects by GNATdashboard.

3.1.1 ChangeLog 24.x

The Sonar Ada Plugin shipped with GNATdashboard now supports **SonarQube** 2025.1 (LTA).

Because of that, it also drops support for old versions of **SonarQube** (*i.e.* all versions *pre* 2025.1).

Follow SonarSource's procedure to update **SonarQube** to a supported version.

SonarQube Scanner

SonarSource replaced **SonarQube Runner** by the new SonarSource Scanner to analyse a project.

GNATdashboard dropped support for **SonarQube Runner**, and is now compatible with **SonarQube Scanner** version 7.0.2.4839, recommended as the default launcher for **SonarQube** 2025.1 (LTA).

Install **SonarQube Scanner** make it available in your PATH prior to executing **GNAThub** on your projects.

AdaCore tools support

GNAT SAS, **GNAT Check** and **GNAT Metric** now better integrate with **SonarQube**:

- a simplified Quality Profile called "GNATdashboard Way" is now the default for Ada projects and supports the latest additions to **GNAT SAS**;
- **GNAT SAS** race condition messages are marked as BUG during the analysis while other messages remain tagged as CODE SMELL to match **SonarQube** jargon;
- suppressed messages are now filtered out of the analysis so that **SonarQube** can mark them as "Resolved";
- **GNAT Metric** metrics are mapped to **SonarQube** metrics when they share the same definition, and custom metrics are better defined to allow **SonarQube** to qualify the trend of a project ("is the quality improving or not?").

3.2 Frequently Asked Questions

3.2.1 SonarQube doesn't report GNAT SAS/GNATcheck rules violations

If SonarQube does not display issues reported by GNAT SAS and/or GNATcheck despite of GNAThub running the tools and collecting the results, make sure all required rules are activated in the Quality Profile you are using for Ada sources.

The SonarQube Ada Plugin that comes with GNATdashboard provides its own quality profile called "GNATdashboard Way". This profile is now the default for Ada projects and supports the latest additions to **GNAT SAS** and **GNAT Check**.

CHAPTER

FOUR

GNATDASHBOARD WEBUI, THE WEB INTERFACE

4.1 WebUI: The GNATdashboard web interface

The GNATdashboard web interface provides a web-based user interface to display, explore and manage the data gathered by GNATdashboard. It works with GNAT SAS, GNATmetric and GNATcheck.

4.1.1 Quick Launch

To launch the web interface, you first need to create the web data files statically.

Creating the data files

- **Gnathub: The creation is done by default when running Gnathub.**

  If you don't want it to be created, use the *--plugins* switch.

  ```
  gnathub -P<project>
  ```

- **GNAT SAS: You need to use the `gnatsas report html` command:**

  ```
  gnatsas report html -P<project>
  ```

4.1.2 Web Interface Overview

Let's take a look at the following picture:

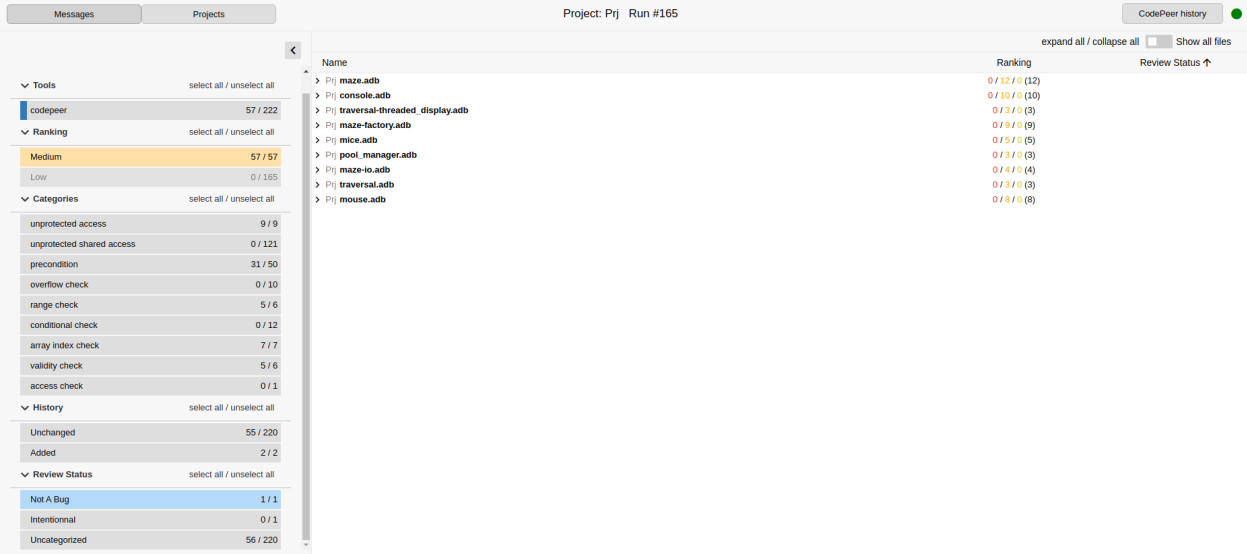
There are three parts in the view: Header, Filter and Content.

The Header and Filter parts are static. The content space will change depending on your navigation.

Header

It is composed of two navigations buttons (on the left side):

- the Message Navigation button, that will lead to *Message navigation* content.
- the Project Navigation button, that will lead to *Project navigation* content.



Filter

This part is collapsible by clicking on the arrow on the upper-right of the filter span.

It will allow you to filter the messages shown in all the views, by clicking on the different span. There are three states for a span:

- activated: the color is bright. You can deactivate it by clicking.
- deactivated: the color is transparent. You can activate it by clicking.
- No occurrence: the number on the right is a 0.

There is some color code that you will retrieve on the content page:

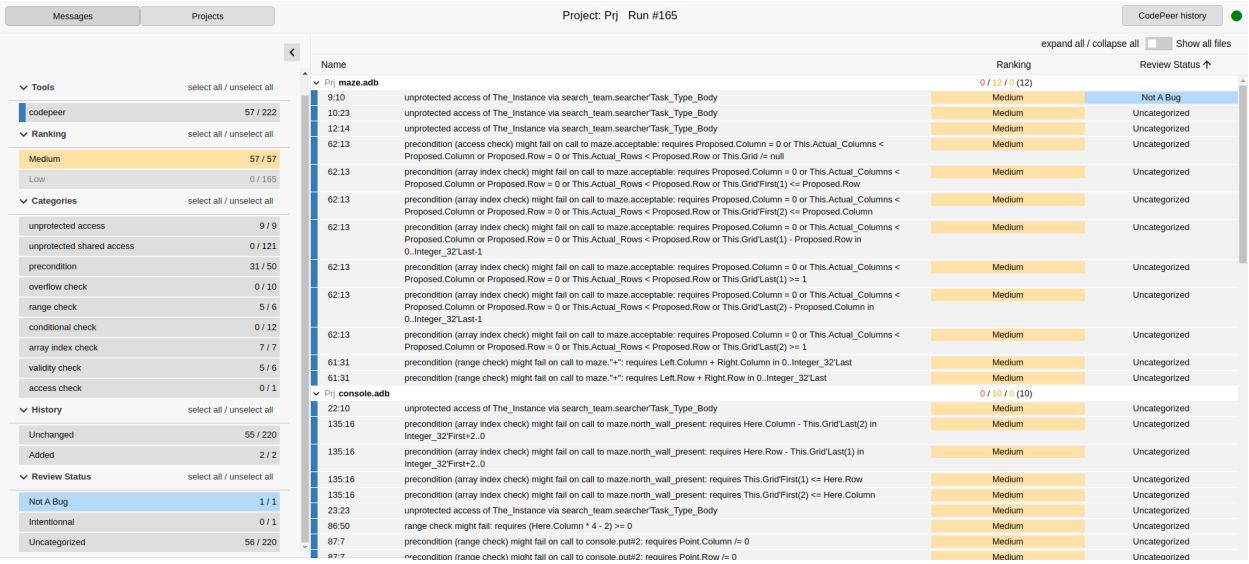
- the left border color representing the tool that reports the message
- the background color (red, orange, yellow, green) for the message ranking.

Content

There are three different pages:

- Message navigation
- Project navigation
- Source view

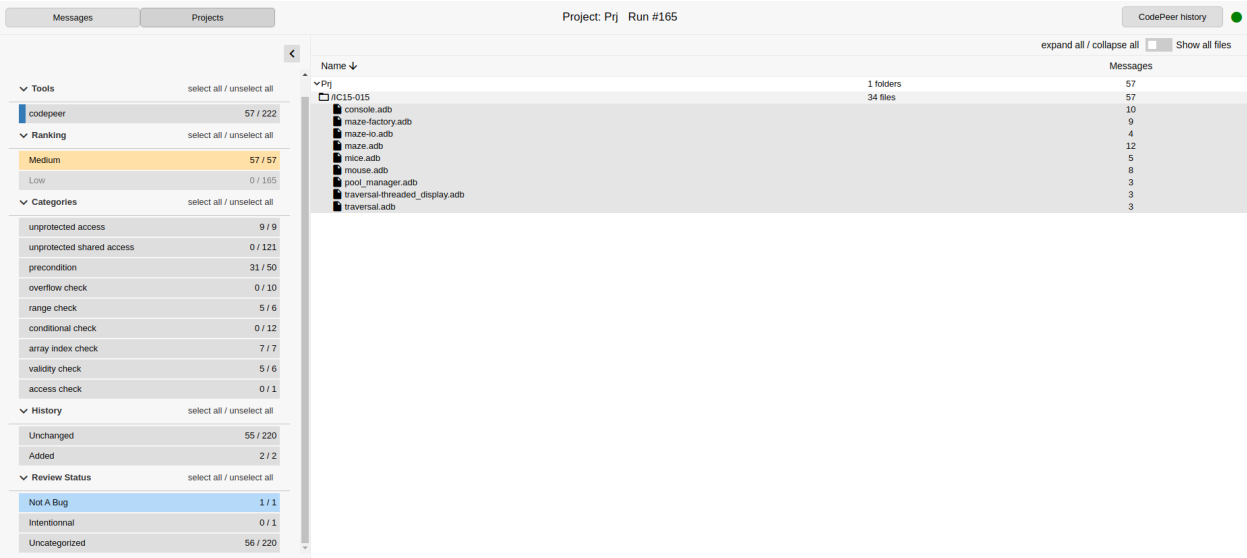
Message navigation



This part allows you to see all the messages, grouped by files.

You can order them by clicking on the table header (Name, Ranking, Message, Review status).

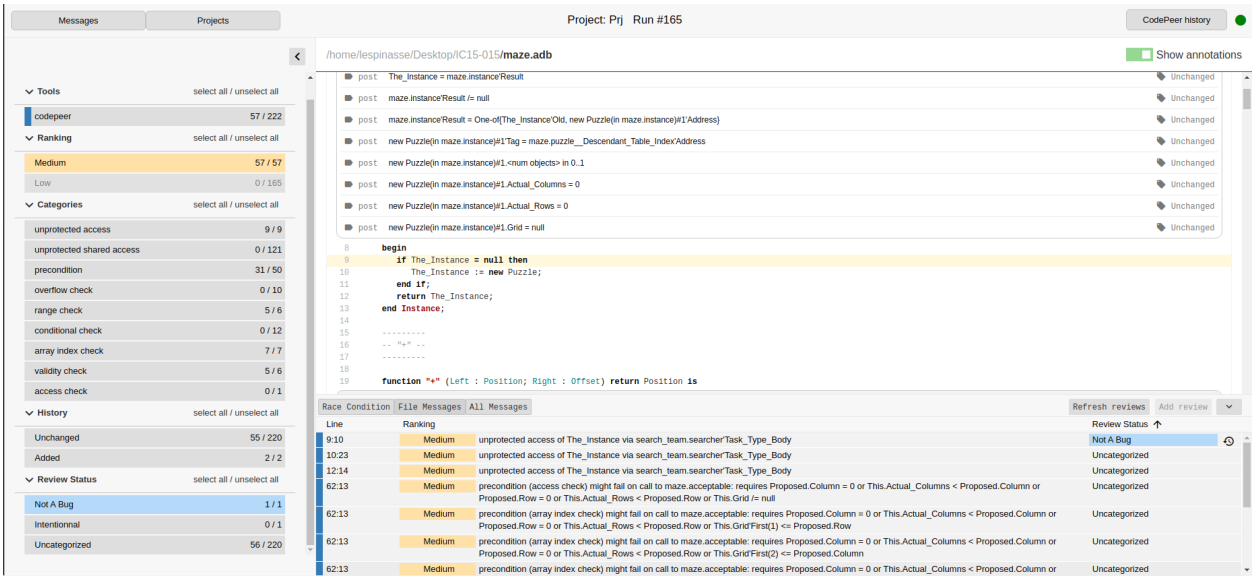
Project navigation



This part allows you to see all the files, grouped by folders, grouped by project. It is a very useful view when looking at a lot of projects.

You can order them by clicking on the table header (Name, Message).

Source view



This page is made of two parts:

- the source file
- the action bar

1. Source file

You can scroll through the source file. If the annotations in GNAT SAS are activated, you will see them in this part.

2. Action bar

This bar is collapsible.

You can see all the message of the current file in the `File messages` tab.

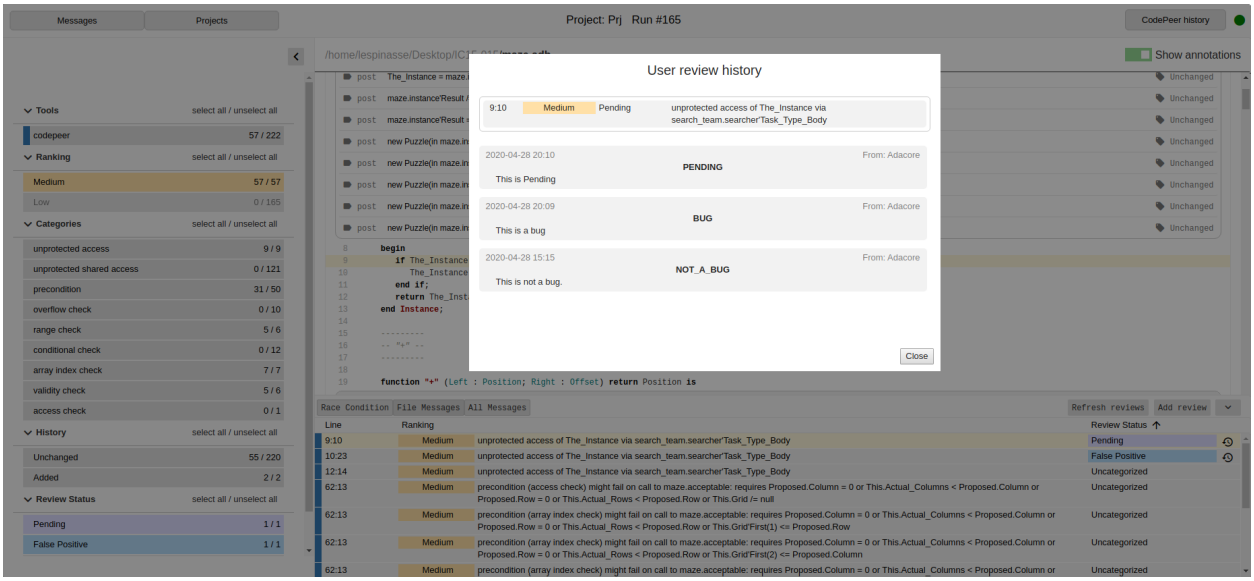
You can see all the messages in the `All messages` tab.

The `line` button, allow you to order message by line.

If GNATmetric is activated, you can see them in the `File Metric` tab. If there are Race condition, you can see them in the `Race condition` tab.

In the `File message` tab you can select a message by clicking on it. This will scroll the source view to the selected line. If there are already some manual reviews made, then you can see the history icon appear, in the rightmost column. By clicking on it, you will see the `User review history` pop-up open:

Note that WebUI used to allow adding reviews, but this functionality was removed in CodePeer 23. Refer to the GNAT SAS User's Guide for more information on how to review messages with GNAT SAS.



4.1.3 Features shortcut

Filter messages

See the *Filter* section.

CHAPTER

FIVE

GNATdashboard DEVELOPER'S MANUAL

5.1 Introduction

GNATdashboard provides an API to access the collected results, and several reporters/exporters.

5.1.1 HTML Report

This reporter generates a standalone static web report.

5.1.2 JSON Report

This format aims at providing an structured output for a tool to consume. It uses the JavaScript Object Notation (JSON) to encode the collected data.

5.2 JSON Entities

5.2.1 Output format

JSON entities are encoded using UTF-8 and can be served with content type *application/json*.

By default reporters output compact JSON, as opposed to pretty-printed JSON, which uses extra whitespace to make the output more readable for humans. Use GNAThub's *debug* switch to generate pretty-printed JSON reports.

Producing (and parsing) the non-pretty compact format is more efficient, so tools can process the content of the reports effectively.

JSON reports can be further gzip compressed to save on local disk space and network transfer time.

5.2.2 CoverageStatus

Coverage status apply to a single line of code.

- *NO_CODE*: this line does not contain or does not generate coverable code;`
- *COVERED*: all statements from this line were covered during tests;
- *NOT_COVERED*: no statements from this line were covered during tests;
- *PARTIALLY_COVERED*: some, but not all, statements from this line were covered during tests.

5.2.3 CoverageInfo

The *CoverageInfo* entity contains detailled information about the coverage results of a line of code.

Field name	Field type	Field description
status	CoverageStatus	The coverage status for this line
hits	number	The number of times this line was executed (if available, *null* otherwise)

CHAPTER

SIX

GNATHUB API REFERENCE

6.1 GNAThub

6.1.1 GNAThub package

Module contents

This module defines the core components of GNAThub plugin mechanism.

It declares module routines and classes implemented in Ada and exported to Python.

In particular the *GNAThub.Plugin* is the base class to use for writing plug-ins.

class GNAThub.**Console**

Bases: `object`

Provide several helper routines implemented in Ada.

static error(*message*, *prefix=None*)

Print an error message.

Always activated.

Parameters

- **message** (*str*) – the message to display
- **prefix** (*str*) – optional prefix to the message

static info(*message*, *prefix=None*)

Print an informative message.

Activated at default verbosity.

Parameters

- **message** (*str*) – the message to display
- **prefix** (*str*) – optional prefix to the message

static ko(*message*, *columns=79*)

Display a KO message.

Execution step ..... [FAILED]

Parameters

- **message** (*str*) – the message to display

- **columns** (*int*) – optional maximum column to use for display (default to 79)

static ok(*message*, *columns=79*)

Display an OK message.

Execution step ..... [PASSED]

Parameters

- **message** (*str*) – the message to display
- **columns** (*int*) – optional maximum column to use for display (default to 79)

static progress(*current*, *total*, *new_line=False*)

Print a progress message.

Activated at default verbosity level. If `new_line` is **True**, then terminates the line with a `n` character.

Parameters

- **current** (*int*) – the current value
- **total** (*int*) – the total value
- **new_line** (*bool*) – whether to terminate with a new line

static set_failure(*message*, *prefix=None*)

Used to set a global run failure.

This is done when one or more plugins have failed during GNAThub run.

Parameters

- **message** (*str*) – the message to display
- **prefix** (*str*) – optional prefix to the message

static warn(*message*, *prefix=None*)

Print a warning message.

Activated at default verbosity output.

Parameters

- **message** (*str*) – the message to display
- **prefix** (*str*) – optional prefix to the message

class GNAThub.**Entity**(*name*, *kind*, *line*, *col_begin*, *col_end*, *resource*)

Bases: `object`

An Entity object, representing an entity in the database.

__init__(*name*, *kind*, *line*, *col_begin*, *col_end*, *resource*)

Return the entity of the given properties, creating it if necessary.

Parameters

- **name** (*str*) – the name of the entity
- **resource** (GNAThub.Resource) – the resource that contains entity

add_messages(*messages*)

Add multiple messages to the given entity.

Prefer this function when there are many messages to insert, for efficiency.

Parameters
messages (*collections.Iterable*) – the messages to add

col_begin

col_end

id

kind

line

static list()

Return all the entities stored in the database.

Returns
the list of all *GNAThub.Entity*

Return type
collections.Iterable[*GNAThub.Entity*]

list_messages()

List all messages associated with this entity.

Returns
a list of *GNAThub.Message*

Return type
collections.Iterable[*GNAThub.Message*]

name

resource_id

exception GNAThub.**Error**

Bases: Exception

Base class for exceptions in this module.

class GNAThub.**Logger**(*name*)

Bases: object

A logger object. Fully implemented in Ada.

__init__(*name*)

Create a new Ada logger object.

From **GNAThub** plug-ins, prefer the use of logging instead of this logger class. A custom logging.Handler is automatically installed to use logging as logging front-end and *GNAThub.Logger* as logging back-end.

Parameters
name (*str*) – the name of the logger

log(*message*)

Log a message.

Parameters
message (*str*) – the message to log

class GNAThub.**Message**(*rule*, *message*, *ranking=1*, *tool_msg_id=0*, *properties=None*)

Bases: object

A Message object, representing one message in the database.

__init__(*rule*, *message*, *ranking=1*, *tool_msg_id=0*, *properties=None*)

Return the message matching the given properties.

Parameters

- **rule** (GNAThub.Rule) – the rule to which this message belongs
- **message** (*str*) – the data to associate to the message: this should be a numeric value if the rule is of METRIC_KIND
- **tool_msg_id** (*int*) – the tool original message id
- **properties** (*collections.Iterable[GNAThub.Property] or None*) – one or more properties

col_begin

col_end

data

id

line

static list()

Return all messages stored in the database.

Returns
the list of all *GNAThub.Message*

Return type
collections.Iterable[*GNAThub.Message*]

ranking

rule_id

tool_msg_id

class GNAThub.**Plugin**

Bases: object

GNAThub plugin interface.

A plugin is a Python class that describe how to configure, run and collect data output by an external tool.

Each plugin should be dedicated to only one tool.

To implement a now plugin, simply creates a new Python class inheriting from this (*GNAThub.Plugin*) abstract base class.

All plugins are collected using the inheritance mechanism, i.e. the GNAThub driver will automatically find all classes implementing the *GNAThub.Plugin* interface. No manual registration needed.

__init__()

Initialize instance properties.

error(*message*, **args*)

Display an error message, prefixed with the plug-in name.

Parameters

- **message** (*str*) – the message to display
- **args** (*collections.Iterable*) – format string arguments

property exec_status

Return the execution status for the tool.

Can be one of the following:

- **GNAThub.NOT_EXECUTED**: plugin did not run yet
- **GNAThub.EXEC_FAILURE**: an error occurred during the plugin execution
- **GNAThub.EXEC_SUCCESS**: the plugin execution completed successfully

Returns
the exit code

Return type
int

info(*message*, **args*)

Display an informative message, prefixed with the plug-in name.

Parameters

- **message** (*str*) – the message to display
- **args** (*collections.Iterable*) – format string arguments

property name

Return the name of the tool.

Returns
the tool name

Return type
str

setup()

Called prior to a call to `Plugin.execute()`.

This is where environment setup should be done to ensure a correct execution of the tool.

teardown()

Called after a call to `Plugin.execute()`.

This is where environment cleanup should be done to ensure a consistent state for a future execution.

warn(*message*, **args*)

Display a warning message, prefixed with the plug-in name.

Parameters

- **message** (*str*) – the message to display
- **args** (*collections.Iterable*) – format string arguments

class GNAThub.**Project**

Bases: object

A project namespace, fully implemented in Ada.

__init__()

Instance constructor.

Do not use directly.

static artifacts_dir()

Return the full path to the root project artifacts directory if defined as IDE attribute in IDE mode, the object directory if is defined, project directory if any of the previous value is set.

Return type
str

static name()

Return the name of the root project.

Return type
str

static object_dir()

Return the full path to the root project object directory.

Return type
str

static object_dirs()

Return the list of object directories in the project tree.

Returns
the list of object directories for the current project

Return type
dict[str, list[str]]

static path()

Return the full path to the root project.

Return type
str

static project_dir()

Return the full path to the root project.

Return type
str

static property_as_list(*key*, *package=None*)

Return a project property.

Returns the list of string representation of the project property from the package GNATdashboard.

Parameters

- **key** (*str*) – the property name
- **package** (*str*) – the package name (default to GNATdashboard package)

Returns
the property value

Return type
list[str]

`static property_as_string(key, package=None)`

Return a project property.

Returns the string representation of the project property from the package GNATdashboard.

Parameters

- **key** (*str*) – the property name
- **package** (*str*) – the package name (default to GNATdashboard package)

Returns
the property value

Return type
str

`static runtime()`

Return the runtime of the root project.

This concerns only the runtime for Ada.

Return type
str

`static scenario_switches()`

Return the scenario as a list of switches of the form -Xvar=value.

Returns
the list of scenario switches passed to GNAThub

Return type
collections.Iterable[str]

`static source_dirs()`

Return the list of source directories for each project.

Returns
the list of all sources directories per project

Return type
dict[str, list[str]]

`static source_file(name)`

Create a new file.

This will automatically try to solve `name` to an absolute path if it currently is a base name. If `name` is an absolute path, it is returned as is. Otherwise, only the base name is used (i.e. we remove any directory information from `name`).

Parameters
name (*str*) – the source file basename

Returns
the full path to the source file

Return type
str

static source_files()

Return the list of source files for each project.

Returns
the list of all sources files per project

Return type
list[str]

static source_suffixes(*language*)

Return a list of Ada source file suffixes.

The list is used by the project manager to find Ada source files, ie. both specifications and implementations.

Parameters
language (*str*) – the language of the sources

Returns
the list of valid Ada source file extensions

Return type
collections.Iterable[str]

static target()

Return the target of the root project.

Return type
str

class GNAThub.**Property**(*identifier*, *name*)

Bases: object

A Property object, corresponding to a message property.

__init__(*identifier*, *name*)

Return a Property, creating it if necessary.

Parameters

- **identifier** (*str*) – the unique identifier of the property
- **name** (*str*) – the display name of the property

id

identifier

static list()

Return all properties stored in the database.

Returns
the list of all *GNAThub.Property*

Return type
collections.Iterable[*GNAThub.Property*]

name

class GNAThub.**Reporter**

Bases: object

Plugin extension point for reporter tools.

abstract report()

Abstract method. Needs custom implementation by derived classes.

Collect the results produced by the external tool. This method is called after `setup()` and before `teardown()`.

Returns
GNAThub.EXEC_SUCCESS on success, GNAThub.EXEC_FAILURE otherwise

Return type
int

class GNAThub.**Resource**(*name*, *kind*)

Bases: `object`

A Resource object, corresponding to a resource in the database.

A resource represents either a file, a directory, or a project.

__init__(*name*, *kind*)

Return a Resource, creating it if necessary.

Parameters

- **name** (*str*) – the name of the resource. For files and directories, this should be a normalized full path: the full path with all links resolved, and with the original filesystem casing. For a project, this is the cased name of the project
- **kind** (*int*) – `PROJECT_KIND`, `DIRECTORY_KIND`, `FILE_KIND` for projects, directories, and files, respectively

add_message(*message*, *line=0*, *col_begin=1*, *col_end=None*)

Add a message to the given resource.

Parameters

- **message** (`GNAThub.Message`) – the Message to add
- **line** (*int*) – the line to associate the message to, if the resource is a file. Use `0` to indicate a message which should be associated to the resource but not to a specific line
- **col_begin** (*int*) – the begin column of the message
- **col_end** (*int*) – the end column of the message. `None` means that the end column should be the same as the begin column.

add_messages(*messages*)

Add multiple messages to the given resource.

Prefer this function when there are many messages to insert, for efficiency. `messages` is a list, each entry being a list of the form:

```
[message, line, col_begin, col_end]
```

`message` is the *GNAThub.Message* to add. `line`, `col_begin`, `col_end`: int, see *add_message()*.

Example:

```
resource.add_messages([
    [ message, 1, 2, 2 ],
    [ message, 2, 1, 1 ],
    [ message, 0, 1, 1 ],
```

(continues on next page)

(continued from previous page)

```
    [ message, 10002, 1, 1 ],
    [ message, 10003, 1, 1 ],
    [ message, 10005, 1, 1 ],
])
```

Parameters
messages (*collections.Iterable*) – the messages to add

static get(*name*)

Return the *GNAThub.Resource* with the given name.

Do not create it if it doesn't exist.

Parameters
name (*str*) – the name of the resource to get

Returns
the *GNAThub.Resource* of that name

Return type
GNAThub.Resource

id

kind

static list()

List all resources stored in the database.

Returns
the list of all *GNAThub.Resource*

Return type
collections.Iterable[*GNAThub.Resource*]

list_entities_messages()

List all entities messages associated with this resource.

Returns
a list of *GNAThub.Message*

Return type
collections.Iterable[*GNAThub.Message*]

list_messages()

List all messages associated with this resource.

Returns
a list of *GNAThub.Message*

Return type
collections.Iterable[*GNAThub.Message*]

name

class GNAThub.**Rule**(*name*, *identifier*, *kind*, *tool*)

Bases: object

A Rule object, representing a rule in the database.

__init__(*name*, *identifier*, *kind*, *tool*)

Return the rule of the given properties, creating it if necessary.

Parameters

- **name** (*str*) – the name of the rule
- **identifier** (*str*) – an unique identifier for this rule (typically, the same as name)
- **kind** (*int*) – RULE_KIND to indicate a rule where messages are given without a numeric value, or METRIC_KIND to indicate a rule where messages correspond to a numeric value
- **tool** (GNAThub.Tool) – the tool that defines this rule

id

identifier

kind

static list()

Return all the rules stored in the database.

Returns
the list of all *GNAThub.Rule*

Return type
collections.Iterable[*GNAThub.Rule*]

name

tool_id

class GNAThub.**Run**(*name*, *argv*, *env=None*, *workdir=None*, *out=None*, *capture_stderr=True*, *append_out=False*)

Bases: object

Class to handle processes.

__init__(*name*, *argv*, *env=None*, *workdir=None*, *out=None*, *capture_stderr=True*, *append_out=False*)

Spawn the process.

Use subprocess.Popen to spawn a process and returns its exit code.

Parameters

- **name** (*str*) – the name of the executable
- **argv** (*collections.Iterable[str]*) – the argument array
- **env** (*dict[str, str]*) – Map containing the environment to pass through to the process. If None, os.environ is used.
- **workdir** (*str*) – the directory in which to execute the process. If None, use the current directory.
- **out** (*str*) – the log file to use
- **append_out** (*bool*) – whether to use 'a' or 'w' flag to open log file
- **capture_stderr** (*bool*) – whether to capture the standard error in the log file

`cmdline_image()`

Return a string image of the given command.

Returns
the command line image

Return type
str

`static expand_argv(name, argv)`

TODO(delay)

Parameters

- **name** (*str*) – the name of the executable
- **argv** (*collections.Iterable[str]*) – the argument array

`output()`

Return the path to the output file.

Returns
the full path to the file

Return type
str

`static quote(arg)`

Return the quoted version of the given argument.

Parameters
arg (*str*) – the argument to quote

Returns
the quoted argument

Return type
str

`wait()`

Wait until process ends and returns its status.

`class GNAThub.Runner`

Bases: `object`

Plugin extension point for analyser tools.

`abstract run()`

Abstract method. Needs custom implementation by derived classes.

Execute the external tool. This method is called after `setup()` and before `teardown()`.

Returns
GNAThub.EXEC_SUCCESS on success, GNAThub.EXEC_FAILURE otherwise

Return type
int

`class GNAThub.Tool(name)`

Bases: `object`

A Tool object, mapping to a Tool entry in the database.

__init__(*name*)

Return the tool of the given name, creating it if necessary.

Parameters
name (*str*) – the name of the tool to create or retrieve

add_messages(*resources_messages*, *entities_messages*)

Add resources and entities (if any) messages to the given tool.

Prefer this function when there are many messages to insert, for efficiency. `resources_messages` is a list, each entry being a list of the form:

```
[resource, message_data]
```

where `message_data` is a list and each entry is of the form

[message, line, col_begin, col_end]

`message` is the *GNAThub.Message* to add. `line`, `col_begin`, `col_end`: int.

Example:

```
tool.add_messages(
        [[resource1, [[ message, 1, 2, 2 ],
                      [ message, 2, 1, 1 ],
                      [ message, 0, 1, 1 ]]],
         [resource2, [[ message, 10002, 1, 1 ],
                      [ message, 10003, 1, 1 ],
                      [ message, 10005, 1, 1 ]]]],
        [[entity1, [[message, 1, 2, 2]]],
         [entity2, [[ message, 1, 2, 2 ],
                    [ message, 2, 1, 1 ]]]])
```

Parameters
messages (*collections.Iterable*) – the messages to add

static clear_references(*tool*)

Clear all references to a tool in the database.

Parameters
tool (*str*) – the name of the tool

id

static list()

List all the tools stored in the database.

Returns
the list of all tools

Return type
collections.Iterable[*GNAThub.Tool*]

name

class GNAThub.**ToolArgsPlaceholder**(*tool_name*)

Bases: `object`

Placeholder for tool-specific arguments.

By default tool-specific arguments (see *tool_args()*) are automatically appended to the end of the command line by *Run*. In some cases this is not flexible enough (eg. when used in combination of the **-output-msg[-only]** switches of **:prog:`codepeer`**. For such cases, this placeholder object should be inserted at the correct position in the command line so that it can be substituted by the appropriate list of arguments.

__init__(*tool_name*)

GNAThub.**collect_project_sources**()

Whether the collect-project-sources switch was passed to the GNAThub driver. A copy of the analyzed project is performed under the gnathub/sonar directory for SonarQube scanner usage.

Returns
whether the collect_project_sources switch was passed or not

Return type
bool

GNAThub.**database**()

Return the path to the GNAThub SQLite database.

Usually <project_object_dir>/gnathub/gnathub.db.

Returns
the full path to the local serialized SQLite database

Return type
str

GNAThub.**db_dir**()

Return the Codepeer DB repository

Returns
the repository path

Return type
str

GNAThub.**dry_run**()

Whether to run in "check mode" or not.

Returns
whether the dry-run flag is enabled or not

Return type
bool

GNAThub.**dry_run_without_project**()

Return true if –dry-run mode without project file.

Returns
bool

GNAThub.**gnatcheck_hide_exempted**()

Whether the gnatcheck-hide-exempted switch was passed to the GNAThub driver or not.

Returns
whether the gnatcheck-hide-exempted switch is passed or not

Return type
bool

GNAThub.**html_data**()

Return the path to the GNAThub-specific directory for HTML report data.

Usually <project_object_dir>/gnathub/html-report/data.

Returns
the full path to the log directory

Return type
str

GNAThub.**incremental**()

Whether the incremental switch was passed to the GNAThub driver or not.

Returns
whether the incremental switch is passed or not

Return type
bool

GNAThub.**jobs**()

Return the number of parallel jobs to execute.

This is the equivalent to using **-j** on the command-line.

Returns
the maximum number of separate processes to spawn

Return type
int

GNAThub.**logs**()

Return the path to the GNAThub-specific directory for logs.

Usually <project_object_dir>/gnathub/logs.

Returns
the full path to the log directory

Return type
str

GNAThub.**output_dir**()

Return the Codepeer output repository

Returns
the repository path

Return type
str

GNAThub.**plugins**()

Return the list of comma-separated plug-in names.

This is the list of plug-ins as specified on the command-line with the **--plugins** switch.

Returns
the list of plug-in name

Return type
collections.Iterable[str]

GNAThub.**port**()

Return the port number provided with the switch.

This is the equivalent to using :*–port* switch on the command-line.

Returns
the port number

Return type
int

GNAThub.**quiet**()

Whether the quiet flag was passed to the GNAThub driver or not.

Returns
whether the quiet flag is enabled or not

Return type
bool

GNAThub.**repositories**()

Return the list of available repositories.

The dictionary contains 3 keys:

- **system**
- **global**
- **local**

These repositories correspond respectively to the `[core]` and `[extra]` directories from the GNAThub installation, and the **Local_Repository** the user can specify in its project file.

Returns
the available repositories details

Return type
dict[str, str]

GNAThub.**root**()

Return the path to the GNAThub-specific root directory.

Usually <project_object_dir>/gnathub.

Returns
the full path to the directory

Return type
str

GNAThub.**sonar_work_dir**()

Return the sonar scanner work dir used by SonarQube provided by the switch.

This is the equivalent to using **--sonar-cache-dir** on the command-line.

Returns
the path of the new sonar sources cache

Return type
str

GNAThub.**subdirs**()

Return the name of the new object directory provided with the switch.

This is the equivalent to using **--subdirs** on the command-line.

Returns
the name of the new object directory

Return type
str

GNAThub.**tool_args**(*tool_name*)

Return the list of extra switches to pass to an inferior tool.

This is the concatenation of switches for the tool `tool_name` as provided on the command-line with the **-targs:** switch.

Parameters
tool_name (*str*) – the name of the tool

Returns
the list of extra switches for `tool_name`

Return type
collections.Iterable[str]

GNAThub.**u_process_all**()

Whether the -U switch was passed to the GNAThub driver or not.

Returns
whether -U switch is enabled or not

Return type
bool

GNAThub.**verbose**()

Whether the verbose flag was passed to the GNAThub driver or not.

Returns
whether the verbosity flag is enabled or not

Return type
bool

6.2 core

6.2.1 GNAThub package

Module contents

This module defines the core components of GNAThub plugin mechanism.

It declares module routines and classes implemented in Ada and exported to Python.

In particular the *GNAThub.Plugin* is the base class to use for writing plug-ins.

class GNAThub.**Console**

Bases: `object`

Provide several helper routines implemented in Ada.

static error(*message*, *prefix=None*)

Print an error message.

Always activated.

Parameters

- **message** (*str*) – the message to display
- **prefix** (*str*) – optional prefix to the message

static info(*message*, *prefix=None*)

Print an informative message.

Activated at default verbosity.

Parameters

- **message** (*str*) – the message to display
- **prefix** (*str*) – optional prefix to the message

static ko(*message*, *columns=79*)

Display a KO message.

Execution step ..... [FAILED]

Parameters

- **message** (*str*) – the message to display
- **columns** (*int*) – optional maximum column to use for display (default to 79)

static ok(*message*, *columns=79*)

Display an OK message.

Execution step ..... [PASSED]

Parameters

- **message** (*str*) – the message to display
- **columns** (*int*) – optional maximum column to use for display (default to 79)

static progress(*current*, *total*, *new_line=False*)

Print a progress message.

Activated at default verbosity level. If `new_line` is **True**, then terminates the line with a `n` character.

Parameters

- **current** (*int*) – the current value
- **total** (*int*) – the total value
- **new_line** (*bool*) – whether to terminate with a new line

static set_failure(*message*, *prefix=None*)

Used to set a global run failure.

This is done when one or more plugins have failed during GNAThub run.

Parameters

- **message** (*str*) – the message to display
- **prefix** (*str*) – optional prefix to the message

static warn(*message*, *prefix=None*)

Print a warning message.

Activated at default verbosity output.

Parameters

- **message** (*str*) – the message to display
- **prefix** (*str*) – optional prefix to the message

class GNAThub.**Entity**(*name*, *kind*, *line*, *col_begin*, *col_end*, *resource*)

Bases: object

An Entity object, representing an entity in the database.

__init__(*name*, *kind*, *line*, *col_begin*, *col_end*, *resource*)

Return the entity of the given properties, creating it if necessary.

Parameters

- **name** (*str*) – the name of the entity
- **resource** (GNAThub.Resource) – the resource that contains entity

add_messages(*messages*)

Add multiple messages to the given entity.

Prefer this function when there are many messages to insert, for efficiency.

Parameters
messages (*collections.Iterable*) – the messages to add

col_begin

col_end

id

kind

line

static list()

Return all the entities stored in the database.

Returns
the list of all *GNAThub.Entity*

Return type
collections.Iterable[*GNAThub.Entity*]

list_messages()

List all messages associated with this entity.

Returns
a list of *GNAThub.Message*

Return type
collections.Iterable[*GNAThub.Message*]

name

resource_id

exception GNAThub.**Error**

Bases: `Exception`

Base class for exceptions in this module.

class GNAThub.**Logger**(*name*)

Bases: `object`

A logger object. Fully implemented in Ada.

__init__(*name*)

Create a new Ada logger object.

From **GNAThub** plug-ins, prefer the use of `logging` instead of this logger class. A custom `logging.Handler` is automatically installed to use `logging` as logging front-end and *GNAThub.Logger* as logging back-end.

Parameters
name (*str*) – the name of the logger

log(*message*)

Log a message.

Parameters
message (*str*) – the message to log

class GNAThub.**Message**(*rule*, *message*, *ranking=1*, *tool_msg_id=0*, *properties=None*)

Bases: `object`

A Message object, representing one message in the database.

__init__(*rule*, *message*, *ranking=1*, *tool_msg_id=0*, *properties=None*)

Return the message matching the given properties.

Parameters

- **rule** (*GNAThub.Rule*) – the rule to which this message belongs
- **message** (*str*) – the data to associate to the message: this should be a numeric value if the rule is of METRIC_KIND
- **tool_msg_id** (*int*) – the tool original message id
- **properties** (*collections.Iterable[GNAThub.Property] or None*) – one or more properties

col_begin

col_end

data

id

line

static list()

Return all messages stored in the database.

Returns
the list of all *GNAThub.Message*

Return type
collections.Iterable[*GNAThub.Message*]

ranking

rule_id

tool_msg_id

class GNAThub.**Plugin**

Bases: `object`

GNAThub plugin interface.

A plugin is a Python class that describe how to configure, run and collect data output by an external tool.

Each plugin should be dedicated to only one tool.

To implement a now plugin, simply creates a new Python class inheriting from this (*GNAThub.Plugin*) abstract base class.

All plugins are collected using the inheritance mechanism, i.e. the GNAThub driver will automatically find all classes implementing the *GNAThub.Plugin* interface. No manual registration needed.

__init__()

Initialize instance properties.

error(*message*, **args*)

Display an error message, prefixed with the plug-in name.

Parameters

- **message** (*str*) – the message to display
- **args** (*collections.Iterable*) – format string arguments

property exec_status

Return the execution status for the tool.

Can be one of the following:

- **GNAThub.NOT_EXECUTED**: plugin did not run yet
- **GNAThub.EXEC_FAILURE**: an error occurred during the plugin execution
- **GNAThub.EXEC_SUCCESS**: the plugin execution completed successfully

Returns
the exit code

Return type
int

info(*message*, **args*)

Display an informative message, prefixed with the plug-in name.

Parameters

- **message** (*str*) – the message to display
- **args** (*collections.Iterable*) – format string arguments

property name

Return the name of the tool.

Returns
the tool name

Return type
str

setup()

Called prior to a call to `Plugin.execute()`.

This is where environment setup should be done to ensure a correct execution of the tool.

teardown()

Called after a call to `Plugin.execute()`.

This is where environment cleanup should be done to ensure a consistent state for a future execution.

warn(*message*, **args*)

Display a warning message, prefixed with the plug-in name.

Parameters

- **message** (*str*) – the message to display
- **args** (*collections.Iterable*) – format string arguments

class GNAThub.**Project**

Bases: `object`

A project namespace, fully implemented in Ada.

__init__()

Instance constructor.

Do not use directly.

static artifacts_dir()

Return the full path to the root project artifacts directory if defined as IDE attribute in IDE mode, the object directory if is defined, project directory if any of the previous value is set.

Return type
str

static name()

Return the name of the root project.

Return type
str

static object_dir()

Return the full path to the root project object directory.

Return type
str

static object_dirs()

Return the list of object directories in the project tree.

Returns
the list of object directories for the current project

Return type
dict[str, list[str]]

`static path()`

Return the full path to the root project.

Return type
str

`static project_dir()`

Return the full path to the root project.

Return type
str

`static property_as_list(key, package=None)`

Return a project property.

Returns the list of string representation of the project property from the package GNATdashboard.

Parameters

- **key** (*str*) – the property name
- **package** (*str*) – the package name (default to GNATdashboard package)

Returns
the property value

Return type
list[str]

`static property_as_string(key, package=None)`

Return a project property.

Returns the string representation of the project property from the package GNATdashboard.

Parameters

- **key** (*str*) – the property name
- **package** (*str*) – the package name (default to GNATdashboard package)

Returns
the property value

Return type
str

`static runtime()`

Return the runtime of the root project.

This concerns only the runtime for Ada.

Return type
str

`static scenario_switches()`

Return the scenario as a list of switches of the form -Xvar=value.

Returns
the list of scenario switches passed to GNAThub

Return type
collections.Iterable[str]

static source_dirs()

Return the list of source directories for each project.

Returns
the list of all sources directories per project

Return type
dict[str, list[str]]

static source_file(*name*)

Create a new file.

This will automatically try to solve name to an absolute path if it currently is a base name. If name is an absolute path, it is returned as is. Otherwise, only the base name is used (i.e. we remove any directory information from name).

Parameters
name (*str*) – the source file basename

Returns
the full path to the source file

Return type
str

static source_files()

Return the list of source files for each project.

Returns
the list of all sources files per project

Return type
list[str]

static source_suffixes(*language*)

Return a list of Ada source file suffixes.

The list is used by the project manager to find Ada source files, ie. both specifications and implementations.

Parameters
language (*str*) – the language of the sources

Returns
the list of valid Ada source file extensions

Return type
collections.Iterable[str]

static target()

Return the target of the root project.

Return type
str

class GNAThub.Property(*identifier*, *name*)

Bases: object

A Property object, corresponding to a message property.

__init__(*identifier*, *name*)

Return a Property, creating it if necessary.

Parameters

- **identifier** (*str*) – the unique identifier of the property
- **name** (*str*) – the display name of the property

id

identifier

static list()

Return all properties stored in the database.

Returns
the list of all *GNAThub.Property*

Return type
collections.Iterable[*GNAThub.Property*]

name

class GNAThub.**Reporter**

Bases: `object`

Plugin extension point for reporter tools.

abstract report()

Abstract method. Needs custom implementation by derived classes.

Collect the results produced by the external tool. This method is called after `setup()` and before `teardown()`.

Returns
GNAThub.EXEC_SUCCESS on success, GNAThub.EXEC_FAILURE otherwise

Return type
int

class GNAThub.**Resource**(*name*, *kind*)

Bases: `object`

A Resource object, corresponding to a resource in the database.

A resource represents either a file, a directory, or a project.

__init__(*name*, *kind*)

Return a Resource, creating it if necessary.

Parameters

- **name** (*str*) – the name of the resource. For files and directories, this should be a normalized full path: the full path with all links resolved, and with the original filesystem casing. For a project, this is the cased name of the project
- **kind** (*int*) – `PROJECT_KIND`, `DIRECTORY_KIND`, `FILE_KIND` for projects, directories, and files, respectively

add_message(*message*, *line=0*, *col_begin=1*, *col_end=None*)

Add a message to the given resource.

Parameters

- **message** (`GNAThub.Message`) – the Message to add
- **line** (*int*) – the line to associate the message to, if the resource is a file. Use `0` to indicate a message which should be associated to the resource but not to a specific line

- **col_begin** (*int*) – the begin column of the message
- **col_end** (*int*) – the end column of the message. None means that the end column should be the same as the begin column.

add_messages(*messages*)

Add multiple messages to the given resource.

Prefer this function when there are many messages to insert, for efficiency. messages is a list, each entry being a list of the form:

```
[message, line, col_begin, col_end]
```

message is the *GNAThub.Message* to add. line, col_begin, col_end: int, see *add_message()*.

Example:

```
resource.add_messages([
    [ message, 1, 2, 2 ],
    [ message, 2, 1, 1 ],
    [ message, 0, 1, 1 ],
    [ message, 10002, 1, 1 ],
    [ message, 10003, 1, 1 ],
    [ message, 10005, 1, 1 ],
])
```

Parameters
messages (*collections.Iterable*) – the messages to add

static get(*name*)

Return the *GNAThub.Resource* with the given name.

Do not create it if it doesn't exist.

Parameters
name (*str*) – the name of the resource to get

Returns
the *GNAThub.Resource* of that name

Return type
GNAThub.Resource

id

kind

static list()

List all resources stored in the database.

Returns
the list of all *GNAThub.Resource*

Return type
collections.Iterable[*GNAThub.Resource*]

list_entities_messages()

List all entities messages associated with this resource.

Returns
a list of *GNAThub.Message*

Return type
collections.Iterable[*GNAThub.Message*]

`list_messages()`

List all messages associated with this resource.

Returns
a list of *GNAThub.Message*

Return type
collections.Iterable[*GNAThub.Message*]

`name`

`class GNAThub.Rule(name, identifier, kind, tool)`

Bases: `object`

A Rule object, representing a rule in the database.

`__init__(name, identifier, kind, tool)`

Return the rule of the given properties, creating it if necessary.

Parameters

- **name** (*str*) – the name of the rule
- **identifier** (*str*) – an unique identifier for this rule (typically, the same as name)
- **kind** (*int*) – RULE_KIND to indicate a rule where messages are given without a numeric value, or METRIC_KIND to indicate a rule where messages correspond to a numeric value
- **tool** (*GNAThub.Tool*) – the tool that defines this rule

`id`

`identifier`

`kind`

`static list()`

Return all the rules stored in the database.

Returns
the list of all *GNAThub.Rule*

Return type
collections.Iterable[*GNAThub.Rule*]

`name`

`tool_id`

`class GNAThub.Run(name, argv, env=None, workdir=None, out=None, capture_stderr=True, append_out=False)`

Bases: `object`

Class to handle processes.

__init__(*name*, *argv*, *env=None*, *workdir=None*, *out=None*, *capture_stderr=True*, *append_out=False*)

Spawn the process.

Use subprocess.Popen to spawn a process and returns its exit code.

Parameters

- **name** (*str*) – the name of the executable
- **argv** (*collections.Iterable[str]*) – the argument array
- **env** (*dict[str, str]*) – Map containing the environment to pass through to the process. If None, os.environ is used.
- **workdir** (*str*) – the directory in which to execute the process. If None, use the current directory.
- **out** (*str*) – the log file to use
- **append_out** (*bool*) – whether to use 'a' or 'w' flag to open log file
- **capture_stderr** (*bool*) – whether to capture the standard error in the log file

cmdline_image()

Return a string image of the given command.

Returns
the command line image

Return type
str

static expand_argv(*name*, *argv*)

TODO(delay)

Parameters

- **name** (*str*) – the name of the executable
- **argv** (*collections.Iterable[str]*) – the argument array

output()

Return the path to the output file.

Returns
the full path to the file

Return type
str

static quote(*arg*)

Return the quoted version of the given argument.

Parameters
arg (*str*) – the argument to quote

Returns
the quoted argument

Return type
str

wait()

Wait until process ends and returns its status.

class GNAThub.**Runner**

Bases: `object`

Plugin extension point for analyser tools.

abstract run()

Abstract method. Needs custom implementation by derived classes.

Execute the external tool. This method is called after `setup()` and before `teardown()`.

Returns
GNAThub.EXEC_SUCCESS on success, GNAThub.EXEC_FAILURE otherwise

Return type
int

class GNAThub.**Tool**(*name*)

Bases: `object`

A Tool object, mapping to a Tool entry in the database.

__init__(*name*)

Return the tool of the given name, creating it if necessary.

Parameters
name (*str*) – the name of the tool to create or retrieve

add_messages(*resources_messages*, *entities_messages*)

Add resources and entities (if any) messages to the given tool.

Prefer this function when there are many messages to insert, for efficiency. `resources_messages` is a list, each entry being a list of the form:

```
[resource, message_data]
```

where `message_data` is a list and each entry is of the form

[message, line, col_begin, col_end]

`message` is the *GNAThub.Message* to add. `line`, `col_begin`, `col_end`: int.

Example:

```
tool.add_messages(
        [[resource1, [[ message, 1, 2, 2 ],
                      [ message, 2, 1, 1 ],
                      [ message, 0, 1, 1 ]]],
         [resource2, [[ message, 10002, 1, 1 ],
                      [ message, 10003, 1, 1 ],
                      [ message, 10005, 1, 1 ]]]],
        [[entity1, [[message, 1, 2, 2]]],
         [entity2, [[ message, 1, 2, 2 ],
                    [ message, 2, 1, 1 ]]]])
```

Parameters
messages (*collections.Iterable*) – the messages to add

static clear_references(*tool*)

Clear all references to a tool in the database.

Parameters
tool (*str*) – the name of the tool

id

static list()

List all the tools stored in the database.

Returns
the list of all tools

Return type
collections.Iterable[*GNAThub.Tool*]

name

class GNAThub.**ToolArgsPlaceholder**(*tool_name*)

Bases: object

Placeholder for tool-specific arguments.

By default tool-specific arguments (see *tool_args()*) are automatically appended to the end of the command line by *Run*. In some cases this is not flexible enough (eg. when used in combination of the **-output-msg[-only]** switches of **:prog:`codepeer`**. For such cases, this placeholder object should be inserted at the correct position in the command line so that it can be substituted by the appropriate list of arguments.

__init__(*tool_name*)

GNAThub.**collect_project_sources**()

Whether the collect-project-sources switch was passed to the GNAThub driver. A copy of the analyzed project is performed under the gnathub/sonar directory for SonarQube scanner usage.

Returns
whether the collect_project_sources switch was passed or not

Return type
bool

GNAThub.**database**()

Return the path to the GNAThub SQLite database.

Usually <project_object_dir>/gnathub/gnathub.db.

Returns
the full path to the local serialized SQLite database

Return type
str

GNAThub.**db_dir**()

Return the Codepeer DB repository

Returns
the repository path

Return type
str

GNAThub.**dry_run**()

Whether to run in "check mode" or not.

Returns
whether the dry-run flag is enabled or not

Return type
bool

GNAThub.**dry_run_without_project**()

Return true if –dry-run mode without project file.

Returns
bool

GNAThub.**gnatcheck_hide_exempted**()

Whether the gnatcheck-hide-exempted switch was passed to the GNAThub driver or not.

Returns
whether the gnatcheck-hide-exempted switch is passed or not

Return type
bool

GNAThub.**html_data**()

Return the path to the GNAThub-specific directory for HTML report data.

Usually <project_object_dir>/gnathub/html-report/data.

Returns
the full path to the log directory

Return type
str

GNAThub.**incremental**()

Whether the incremental switch was passed to the GNAThub driver or not.

Returns
whether the incremental switch is passed or not

Return type
bool

GNAThub.**jobs**()

Return the number of parallel jobs to execute.

This is the equivalent to using **-j** on the command-line.

Returns
the maximum number of separate processes to spawn

Return type
int

GNAThub.**logs**()

Return the path to the GNAThub-specific directory for logs.

Usually <project_object_dir>/gnathub/logs.

Returns
the full path to the log directory

Return type
str

GNAThub.**output_dir**()

Return the Codepeer output repository

Returns
the repository path

Return type
str

GNAThub.**plugins**()

Return the list of comma-separated plug-in names.

This is the list of plug-ins as specified on the command-line with the **--plugins** switch.

Returns
the list of plug-in name

Return type
collections.Iterable[str]

GNAThub.**port**()

Return the port number provided with the switch.

This is the equivalent to using :*–port* switch on the command-line.

Returns
the port number

Return type
int

GNAThub.**quiet**()

Whether the quiet flag was passed to the GNAThub driver or not.

Returns
whether the quiet flag is enabled or not

Return type
bool

GNAThub.**repositories**()

Return the list of available repositories.

The dictionary contains 3 keys:

- **system**
- **global**
- **local**

These repositories correspond respectively to the [core] and [extra] directories from the GNAThub installation, and the **Local_Repository** the user can specify in its project file.

Returns
the available repositories details

Return type
dict[str, str]

GNAThub.**root**()

Return the path to the GNAThub-specific root directory.

Usually <project_object_dir>/gnathub.

Returns
the full path to the directory

Return type
str

GNAThub.**sonar_work_dir**()

Return the sonar scanner work dir used by SonarQube provided by the switch.

This is the equivalent to using **--sonar-cache-dir** on the command-line.

Returns
the path of the new sonar sources cache

Return type
str

GNAThub.**subdirs**()

Return the name of the new object directory provided with the switch.

This is the equivalent to using **--subdirs** on the command-line.

Returns
the name of the new object directory

Return type
str

GNAThub.**tool_args**(*tool_name*)

Return the list of extra switches to pass to an inferior tool.

This is the concatenation of switches for the tool tool_name as provided on the command-line with the **-targs:** switch.

Parameters
tool_name (*str*) – the name of the tool

Returns
the list of extra switches for tool_name

Return type
collections.Iterable[str]

GNAThub.**u_process_all**()

Whether the -U switch was passed to the GNAThub driver or not.

Returns
whether -U switch is enabled or not

Return type
bool

GNAThub.**verbose**()

Whether the verbose flag was passed to the GNAThub driver or not.

Returns
whether the verbosity flag is enabled or not

Return type
bool

6.2.2 codepeer module

GNAThub plug-in for the CodePeer command-line tool.

It exports the CodePeer class which implements the *GNAThub.Plugin* interface. This allows GNAThub's plug-in scanner to automatically find this module and load it as part of the GNAThub default execution.

class codepeer.**CodePeer**

Bases: *Plugin*, *Runner*, *Reporter*

CodePeer plugin for GNAThub.

Configures and executes CodePeer, then analyzes the output.

CODEPEER_TO_RANKING = {'annotation': 0, 'high': 5, 'info': 2, 'low': 3, 'medium': 4}

__init__()

Initialize instance properties.

property output_dir

Return the path to the directory where to generate the Codepeer files

Returns
the full path to the output directory

Return type
str

report()

Execute CodePeer message reader and parses the output.

Sets the exec_status property according to the success of the analysis:

- GNAThub.EXEC_SUCCESS: on successful execution and analysis
- GNAThub.EXEC_FAILURE: on any error

run()

Execute CodePeer.

Sets the exec_status property according to the success of the execution of the tool:

- GNAThub.EXEC_SUCCESS: on successful execution
- GNAThub.EXEC_FAILURE: on any error

6.2.3 gcov module

GNAThub plug-in for the Gcov command-line tool.

It exports the Gcov class which implements the *GNAThub.Plugin* interface. This allows GNAThub's plug-in scanner to automatically find this module and load it as part of the GNAThub default execution.

class gcov.**Gcov**

Bases: *Plugin*, *Reporter*

Gcov plugin for GNAThub.

Retrieves .gcov generated files from the project root object directory, parses them and feeds the database with the data collected from each files.

GCOV_EXT = '.gcov'

__init__()

Initialize instance properties.

report()

Analyse the report files generated by **Gcov**.

Finds all .gcov files in the object directory and parses them.

Sets the exec_status property according to the success of the analysis:

- GNAThub.EXEC_SUCCESS: on successful execution and analysis
- GNAThub.EXEC_FAILURE: on any error

6.2.4 gnatcheck module

GNAThub plug-in for the GNATcheck command-line tool.

It exports the GNATcheck class which implements the *GNAThub.Plugin* interface. This allows GNAThub's plug-in scanner to automatically find this module and load it as part of the GNAThub default execution.

class gnatcheck.**GNATcheck**

Bases: *Plugin*, *Runner*, *Reporter*

GNATcheck plugin for GNAThub.

Configures and executes GNATcheck, then analyzes the output.

VALID_EXIT_CODES = (0, 1)

__init__()

Initialize instance properties.

report()

Parse GNATcheck output file report.

Returns according to the success of the analysis:

- GNAThub.EXEC_SUCCESS: on successful execution and analysis
- GNAThub.EXEC_FAILURE: on any error

Identify two type of messages with different format:

- basic message

- message for package instantiation

run()

Execute GNATcheck.

Returns according to the success of the execution of the tool:

- `GNAThub.EXEC_SUCCESS`: on successful execution
- `GNAThub.EXEC_FAILURE`: on any error

6.2.5 gnatcoverage module

GNATdashboard plug-in for the GNATcoverage command-line tool.

It exports the GNATcoverage class which implements the *GNAThub.Plugin* interface. This allows GNATdashboard's plug-in scanner to automatically find this module and load it as part of the GNATdashboard default execution.

class gnatcoverage.**GNATcoverage**

Bases: *Plugin*, *Reporter*

GNATcoverage plugin for GNATdashboard.

Retrieves .xcov generated files from the project root object directory, parses them and feeds the database with the data collected from each files.

__init__()

Initialize instance properties.

report()

Analyse the report files generated by **GNATcoverage**.

Finds all .xcov files in the object directory and parses them.

Sets the exec_status property according to the success of the analysis:

- `GNAThub.EXEC_SUCCESS`: on successful execution and analysis
- `GNAThub.EXEC_FAILURE`: on any error

6.2.6 gnatmetric module

GNATdashboard plug-in for the GNATmetric and LALmetric command-line tool.

It exports the GNATmetric class which implements the *GNAThub.Plugin* interface. This allows GNATdashboard's plug-in scanner to automatically find this module and load it as part of the GNATdashboard default execution.

class gnatmetric.**GNATmetric**

Bases: *Plugin*, *Runner*, *Reporter*

GNATmetric & LALmetric plugin for GNATdashboard.

RANKING = 2

VALID_EXIT_CODES = (0, 1)

__init__()

Initialize instance properties.

property name

Return the name of the tool.

Returns
the tool name

Return type
str

parse_config(*tree*)

Parse the config block to retrieve the names to be displayed for each rule.

parse_metrics(*node*, *entity=False*)

Parse the xml *node* returns a list of metrics

parse_units(*node*, *resource*)

Recursively parse the unit node until all of them are found

report()

Parse GNATmetric XML report and save data to the database.

Returns according to the success of the analysis:

- `GNAThub.EXEC_SUCCESS`: transactions committed to database
- `GNAThub.EXEC_FAILURE`: error while parsing the xml report

run()

Execute GNATmetric.

Returns according to the success of the execution of the tool:

- `GNAThub.EXEC_SUCCESS`: on successful execution
- `GNAThub.EXEC_FAILURE`: on any error

6.2.7 gnatstack module

GNAThub plug-in for the GNATstack command-line tool.

It exports the GNATstack class which implements the *GNAThub.Plugin* interface. This allows GNAThub's plug-in scanner to automatically find this module and load it as part of the GNAThub default execution.

class gnatstack.**GNATstack**

Bases: *Plugin*, *Runner*, *Reporter*

GNATstack plugin for GNAThub.

RANKING = 2

VALID_EXIT_CODES = (0, 1)

__init__()

Initialize instance properties.

pp_msg(*id*, *msg*)

pp_name(*name*)

report()

Parse GNATstack output file report.

Returns according to the success of the analysis:

- GNAThub.EXEC_SUCCESS: on successful execution and analysis
- GNAThub.EXEC_FAILURE: on any error

run()

Execute GNATstack.

Returns according to the success of the execution of the tool:

- GNAThub.EXEC_SUCCESS: on successful execution
- GNAThub.EXEC_FAILURE: on any error

6.2.8 html_report module

GNAThub plug-in for the generation of a standalone rich HTML report.

It exports the HTMLReport class which implements the *GNAThub.Plugin* interface. This allows GNAThub's plug-in scanner to automatically find this module and load it as part of the GNAThub default execution.

class html_report.**HTMLReport**

Bases: *Plugin*, *Reporter*

HTMLReport plugin for GNAThub.

copy_contents(*src*, *dstfile*)

property name

Return the name of the tool.

Returns
the tool name

Return type
str

property output_dir

Return the path to the directory where to generate the HTML report.

Returns
the full path to the output directory

Return type
str

output_json(*src*, *dst*)

output_xml(*src*, *dst*)

report()

Generate JSON-encoded representation of the data collected.

verbose_info(*message*)

property webapp_dir

Return the path to the webapp directory.

This directory contains the generic parts of the web application.

Return type
str

6.2.9 sonar_config module

GNAThub plug-in for the generation of SonarQube Scanner configuration file.

It exports the SonarConfig class which implements the *GNAThub.Plugin* interface. This allows GNAThub's plug-in scanner to automatically find this module and load it as part of the GNAThub default execution.

class sonar_config.**SonarConfig**

Bases: *Plugin*, *Runner*

SonarConfig plugin for GNAThub.

property name

Return the name of the tool.

Returns
the tool name

Return type
str

run()

Generate SonarQube Scanner configuration file.

setup()

Called prior to a call to `Plugin.execute()`.

This is where environment setup should be done to ensure a correct execution of the tool.

6.2.10 sonar_scanner module

GNAThub plug-in for the SonarQube Scanner command-line tool.

It exports the SonarScanner class which implements the *GNAThub.Plugin* interface. This allows GNAThub's plug-in runner to automatically find this module and load it as part of the GNAThub default execution.

class sonar_scanner.**SonarScanner**

Bases: *Plugin*, *Reporter*

SonarQube Scanner plugin for GNAThub.

__init__()

Initialize instance properties.

property name

Return the name of the tool.

Returns
the tool name

Return type
str

report()

Execute the SonarQube Scanner.

Returns according to the successful of the analysis:

- GNAThub.EXEC_SUCCESS: on successful execution and analysis
- GNAThub.EXEC_FAILURE: on any error

setup()

Called prior to a call to Plugin.execute().

This is where environment setup should be done to ensure a correct execution of the tool.

6.2.11 spark2014 module

GNAThub plug-in for the SPARK2014 command-line tool.

It exports the SPARK2014 class which implements the *GNAThub.Plugin* interface. This allows GNAThub's plug-in scanner to automatically find this module and load it as part of the GNAThub default execution.

class spark2014.**SPARK2014**

Bases: *Plugin*, *Runner*, *Reporter*

SPARK2014 plugin for GNAThub.

Configures and executes GNATprove, then analyzes the output.

SPARK_TO_RANKING = {'high': 5, 'info': 2, 'low': 3, 'medium': 4, 'warning': 4}

__init__()

Initialize instance properties.

report()

Execute GNATprove message reader and parses the output.

Sets the exec_status property according to the success of the analysis:

- GNAThub.EXEC_SUCCESS: on successful execution and analysis
- GNAThub.EXEC_FAILURE: on any error

run()

Execute GNATprove.

Sets the exec_status property according to the success of the execution of the tool:

- GNAThub.EXEC_SUCCESS: on successful execution
- GNAThub.EXEC_FAILURE: on any error

PYTHON MODULE INDEX

INDEX

Symbols

A

C

D

E

G

H

I

J

K

L

M

N

O

P