aboutcode

AboutCode.org authors and contributors

Jan 18, 2024

CONTENTS

Welcome to the AboutCode documentation homepage. AboutCode.org is a community of open source developers who are trying to make open source easier to use by providing open source tools to discover, identify and track open source components (aka Software Composition Analysis – SCA). AboutCode is the collective name for these open source tools.

This home page provides a directory of our major projects and their documentation on ReadTheDocs plus information about how to contribute to AboutCode documentation.

All community contributions are welcome.

OVERVIEW

1.1 AboutCode Project Overview

The primary current AboutCode projects are:

- ScanCode Toolkit: is a code scanning "engine" and command-line tool to detect the provenance and license of code and its dependencies. ScanCode TK is a command-line tool with many scanning options and output formats (JSON, HTML, CSV or SPDX). ScanCode detects licenses, copyrights, package manifests and more in both source code and binary files. There is already a large set of pre- and post-plugins and you can also create your own plugins.

  - Read more at: https://scancode-toolkit.readthedocs.io

  - Get the code at: https://github.com/nexB/scancode-toolkit

- ScanCode.io: is a Cloud application server that automates Software Composition Analysis with pipelines. ScanCode.io has standard pipelines for Docker and VM images, root filesystems and packages. ScanCode.io organizes these complex code analyses as scripted pipelines and stores the results in its database for automated code analysis.

  - Read more at: https://scancodeio.readthedocs.io

  - Get the code at: https://github.com/nexB/scancode.io

- ScanCode Workbench: is a desktop application (based on Electron) to review the results of a scan and document your conclusions about the origin and license of software components and packages.

  - Read more at: https://scancode-workbench.readthedocs.io

  - Get the code at: https://github.com/nexB/scancode-workbench

- VulnerableCode: is an early stage project to provide a free and open source database of vulnerabilities and the packages they impact with tools to aggregate and correlate those vulnerabilities. The initial development of VulnerableCode was supported by the NLNet Foundation.

- **–** Read more at: https://vulnerablecode.readthedocs.io

- **–** Get the code at: https://github.com/nexB/vulnerablecode

- AboutCode Toolkit: AboutCode Toolkit provides a set of command-line tools to generate Attribution documents and software BOM reports from any source. It also provides a standard ABCD format for yaml files to document software provenance and license metadata in a codebase where this is not already covered by a package manager.

  - **–** Read more at: https://aboutcode-toolkit.readthedocs.io

  - **–** Get the code at: https://github.com/nexB/aboutcode-toolkit

CONTRIBUTING AND GSOC/GSOD

2.1 Contributing to AboutCode

The AboutCode organization welcomes you and your interest in contributing to our organization. We are always looking for enthusiatic contributors and we are willing to lend a helping hand if you have any questions or comments. That being said, here a few resources to help you get started.

1) **Take a look through our public repos here: https://github.com/nexB/**

    • Find one you are interested in and check out its open **Issues**

2) **If you have specific questions browse through our documentation here:**
   https://aboutcode.readthedocs.io/en/latest/

    • Depending on the project, there may be a seperate ReadTheDocs website

    • If you did not find what you were looking for or you still have questions, open an issue on the relevant repository, or ask directly via Gitter

You can always interact with the AboutCode organization and our extended community on Gitter

2.1.1 Contribution Guidelines:

Writing good Commit Messages

What is good commit message? We want to avoid this: https://xkcd.com/1296/

Read these articles:

   • by @cbeams How to Write a Git Commit Message

   • this README from Linus Torvalds https://github.com/torvalds/subsurface-for-dirk/blob/0f58510ce0244513521296b75281fcc32f72a931/README#L73

   • from the Git book: https://git-scm.com/book/en/v2/Distributed-Git-Contributing-to-a-Project

The main style points are these:

Subject:

   • Add a issue number at the end of the line when available as in "#234"

   • Limit the subject line to 50 characters

   • Capitalize the subject line

   • Do not end the subject line with a period

   • Use the imperative mood in the subject line: you are giving orders to the codebase

Body:

- Separate subject from body with a blank line

- Wrap the body at 72 characters. Use only spaces for formatting, not tabs.

- Use the body to explain what and why vs. how

- use bullets with a * if needed

- Add a Reported-by: if needed

- End your message with a Signed-off-by: prefixed by a blank line

Other comments:

We like to suffix the subject line with an issue number. If this was a trivial change it may not have one though. If it had one a you would use #156 as a suffix to the first line.

We like to tell why the commit is there and use an imperative style, like if you were giving an order to the codebase with your commit:

e.g rather than : `Minor fix for unnecessary operations.` may be `Remove unnecessary operations #123` or:

```
Remove unnecessary operations #123

    * If the ts timestamp does not exist, do not compare with old one.
```

You need to add a signoff to your commit. So the final message would have looked like this:

```
Remove unnecessary operations #123

    * If the ts timestamp does not exist, do not compare with old one.

Signed-off-by: Philippe Ombredanne <pombredanne@nexb.com>
```

Testing

ToDo

We need to create this content.

Contributing to the Documentation

Setup Local Build

To get started, create or identify a working directory on your local machine.

Open that directory and execute the following command in a terminal session:

```
git clone https://github.com/nexB/aboutcode.git
```

That will create an /aboutcode directory in your working directory. Now you can install the dependencies in a virtualenv:

```
cd aboutcode
./configure --docs
```

Note: In case of windows, run `configure --docs` instead of this.

Now, this will install the following prerequisites:

- Sphinx

- sphinx_rtd_theme (the format theme used by ReadTheDocs)

- docs8 (style linter)

These requirements are already present in setup.cfg and *./configure –docs* installs them.

Now you can build the HTML documents locally:

```
source venv/bin/activate
cd docs
make html
```

Assuming that your Sphinx installation was successful, Sphinx should build a local instance of the documentation .html files:

```
open build/html/index.html
```

Note: In case this command did not work, for example on Ubuntu 18.04 you may get a message like "Couldn't get a file descriptor referring to the console", try:

```
see build/html/index.html
```

You now have a local build of the AboutCode documents.

Share Document Improvements

Ensure that you have the latest files:

```
git pull
git status
```

Before commiting changes run Continious Integration Scripts locally to run tests. Refer *Continuous Integration* for instructions on the same.

Follow standard git procedures to upload your new and modified files. The following commands are examples:

```
git status
git add source/index.rst
git add source/how-to-scan.rst
git status
git commit -m "New how-to document that explains how to scan"
git status
git push
git status
```

The Scancode-Toolkit webhook with ReadTheDocs should rebuild the documentation after your Pull Request is Merged.

Refer the Pro Git Book available online for Git tutorials covering more complex topics on Branching, Merging, Rebasing etc.

Continuous Integration

The documentations are checked on every new commit through Travis-CI, so that common errors are avoided and documentation standards are enforced. Travis-CI presently checks for these 3 aspects of the documentation :

1. Successful Builds (By using `sphinx-build`)

2. No Broken Links (By Using `link-check`)

3. Linting Errors (By Using `Doc8`)

So run these scripts at your local system before creating a Pull Request:

```
cd docs
./scripts/sphinx_build_link_check.sh
./scripts/doc8_style_check.sh
```

If you don't have permission to run the scripts, run:

```
chmod u+x ./scripts/doc8_style_check.sh
```

Style Checks Using `Doc8`

How To Run Style Tests

In the project root, run the following commands:

```
$ cd docs
$ ./scripts/doc8_style_check.sh
```

A sample output is:

```
Scanning...
Validating...
docs/source/misc/licence_policy_plugin.rst:37: D002 Trailing whitespace
docs/source/misc/faq.rst:45: D003 Tabulation used for indentation
docs/source/misc/faq.rst:9: D001 Line too long
docs/source/misc/support.rst:6: D005 No newline at end of file
========
Total files scanned = 34
Total files ignored = 0
Total accumulated errors = 326
Detailed error counts:
    - CheckCarriageReturn = 0
    - CheckIndentationNoTab = 75
    - CheckMaxLineLength = 190
    - CheckNewlineEndOfFile = 13
```

(continues on next page)

```
- CheckTrailingWhitespace = 47
- CheckValidity = 1
```

Now fix the errors and run again till there isn't any style error in the documentation.

What is Checked?

PyCQA is an Organization for code quality tools (and plugins) for the Python programming language. Doc8 is a sub-project of the same Organization. Refer this README for more details.

What is checked:

- invalid rst format - D000

- lines should not be longer than 100 characters - D001

    - RST exception: line with no whitespace except in the beginning

    - RST exception: lines with http or https URLs

    - RST exception: literal blocks

    - RST exception: rst target directives

- no trailing whitespace - D002

- no tabulation for indentation - D003

- no carriage returns (use UNIX newlines) - D004

- no newline at end of file - D005

Interspinx

ScanCode toolkit documentation uses Intersphinx to link to other Sphinx Documentations, to maintain links to other Aboutcode Projects.

To link sections in the same documentation, standart reST labels are used. Refer Cross-Referencing for more information.

For example:

```
.. _my-reference-label:

Section to cross-reference
--------------------------

This is the text of the section.

It refers to the section itself, see :ref:`my-reference-label`.
```

Now, using Intersphinx, you can create these labels in one Sphinx Documentation and then referance these labels from another Sphinx Documentation, hosted in different locations.

You just have to add the following in the `conf.py` file for your Sphinx Documentation, where you want to add the links:

```
extensions = [
'sphinx.ext.intersphinx'
]

intersphinx_mapping = {'aboutcode': ('https://aboutcode.readthedocs.io/en/latest/',␣
↪None)}
```

To show all Intersphinx links and their targets of an Intersphinx mapping file, run:

```
python -msphinx.ext.intersphinx https://aboutcode.readthedocs.io/en/latest/objects.inv
```

> **Warning:** `python -msphinx.ext.intersphinx https://aboutcode.readthedocs.io/objects.inv`
> will give error.

This enables you to create links to the `aboutcode` Documentation in your own Documentation, where you modified the configuration file. Links can be added like this:

```
For more details refer :ref:`aboutcode:doc_style_guide`.
```

You can also not use the `aboutcode` label assigned to all links from aboutcode.readthedocs.io, if you don't have a label having the same name in your Sphinx Documentation. Example:

```
For more details refer :ref:`doc_style_guide`.
```

If you have a label in your documentation which is also present in the documentation linked by Intersphinx, and you link to that label, it will create a link to the local label.

For more information, refer this tutorial named Using Intersphinx.

Style Conventions for the Documentaion

1. Headings

   (Refer) Normally, there are no heading levels assigned to certain characters as the structure is determined from the succession of headings. However, this convention is used in Python's Style Guide for documenting which you may follow:

   # with overline, for parts

   - with overline, for chapters

   =, for sections

   -, for subsections

   ^, for sub-subsections

   ", for paragraphs

2. Heading Underlines

   Do not use underlines that are longer/shorter than the title headline itself. As in:

   ```
   Correct :
   ```

```
Extra Style Checks
------------------

Incorrect :

Extra Style Checks
----------------------
```

Note: Underlines shorter than the Title text generates Errors on sphinx-build.

3. Internal Links

   Using `:ref:` is advised over standard reStructuredText links to sections (like `` `Section title`_ ``) because it works across files, when section headings are changed, will raise warnings if incorrect, and works for all builders that support cross-references. However, external links are created by using the standard `` `Section title`_ `` method.

4. Eliminate Redundancy

   If a section/file has to be repeated somewhere else, do not write the exact same section/file twice. Use `.. include:  ../README.rst` instead. Here, `../` refers to the documentation root, so file location can be used accordingly. This enables us to link documents from other upstream folders.

5. Using `:ref:` only when necessary

   Use `:ref:` to create internal links only when needed, i.e. it is referenced somewhere. Do not create references for all the sections and then only reference some of them, because this created unnecessary references. This also generates ERROR in `restructuredtext-lint`.

6. Spelling

   You should check for spelling errors before you push changes. Aspell is a GNU project Command Line tool you can use for this purpose. Download and install Aspell, then execute `aspell check <file-name>` for all the files changed. Be careful about not changing commands or other stuff as Aspell gives prompts for a lot of them. Also delete the temporary `.bak` files generated. Refer the manual for more information on how to use.

7. Notes and Warning Snippets

   Every `Note` and `Warning` sections are to be kept in `rst_snippets/note_snippets/` and `rst_snippets/warning_snippets/` and then included to eliminate redundancy, as these are frequently used in multiple files.

Converting from Markdown

If you want to convert a `.md` file to a `.rst` file, this tool does it pretty well. You'd still have to clean up and check for errors as this contains a lot of bugs. But this is definitely better than converting everything by yourself.

This will be helpful in converting GitHub wiki's (Markdown Files) to reStructuredtext files for Sphinx/ReadTheDocs hosting.

2.2 GSoC – Google Summer of Code

Google Summer of Code is a global annual program focused on introducing students to open source software development. GSoC is completely online designed to encourage university student participation in open source software development. It was started by Google in 2005. More about GSoc - <https://summerofcode.withgoogle.com/about/>_

2.2.1 GSoC 2022

Extending license detection to use licenses external to ScanCode Toolkit

Organization: AboutCode
Project: Scancode Toolkit
Mentee: Kevin Ji (KevinJi22)
Mentors: Philippe Ombredanne, AyanSinhaMahapatra, Jono Yang

Overview

When doing license detection, ScanCode uses the licenses and rules in the ScanCode LicenseDB. The goal of this project is to extend the capabilities of ScanCode license detection to include licenses that are external to LicenseDB, such as proprietary licenses to be kept within an organization. I also extended it to include licenses installed from external sources.

Implementation

All the work I did is contained in this single PR. I added a new command line option called `--additional-license-directory` that someone can use to include additional licenses/rules contained in other directories in the license index. Scancode Toolkit uses this license index when doing license detection. This option must be called with `--reindex-licenses` to explicitly regenerate the license cache, and then when doing license scans, users can just use the regular `--license` option and these additional licenses and/or rules will be used in license detection.

This change also allows users to install directories of licenses or rules to their local machine, and then Scancode Toolkit will detect and include them in the license cache when someone is reindexing the licenses. If someone wants to create a directory of licenses or rules that they want to install and use in Scancode Toolkit, they must subclass a new Plugin class I added. This allows Scancode Toolkit to identify the location of these installed licenses/rules through a unique entry point and add them to the license index.

Finally, all these changes are tested through multiple unit tests validating both correct behavior and error handling as needed.

Post GSoC

I would like to merge this PR into Scancode Toolkit, hopefully allowing users to leverage this feature to expand their license detection capabilities.

Links

Project idea

Official GSoC project page

GSoC Proposal

Documentation page about the feature

Acknowledgements

Thanks to Jono and Phillippe for being my mentors. I enjoyed all the meetings, code reviews, and design discussions. Thank you for your time and your patience!

Create a web application to scan and review a single license text

Organization: AboutCode
Project: Scancode.io
Mentee: Lali Akhil Raj (lf32)
Mentors: Philippe Ombredanne, Thomas Druez, AyanSinhaMahapatra, AvishrantsSh
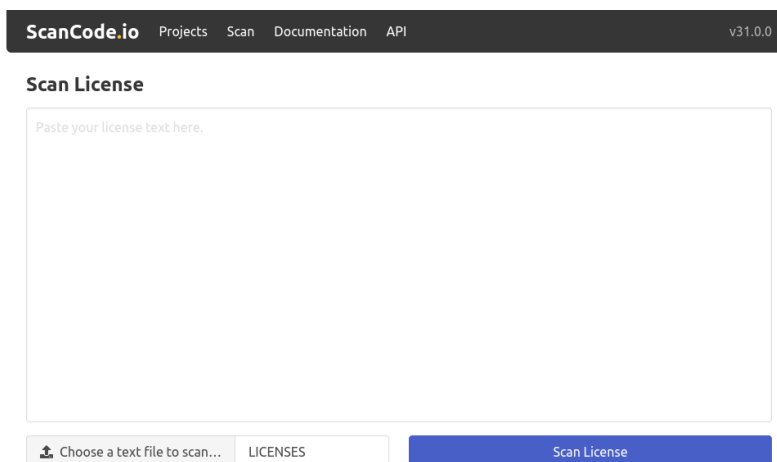
Overview

The aim of this project is to create a Django-based web application (scantext) to scan and review a single license text inside ScanCode.io. The application will be designed around a Django_REST-framework, and It will also allow the integrated reporting of license detection issues in the app based on the results.

Implementation

After the application is created it should have these similar features

- A form to upload/paste text from which scantext

By default, it prefers to scan input text, though both are provided.

- A license summary page to view the analysed results in detail

---

This page is the backbone of the entire project. It is to show everything from highlighting matched licence text to showing details of every match. There are still things that need to be implemented. For now, this is how it looks. On hovering over the dropdown in the left sidebar panel, it's going to show details of their respective match. These are just a few details at the top level. If you click on "view more details," it's going to show more details of that match. Isn't it awesome? I had terrible issues when I was designing it the other way.

Thanks to my mentors for giving me ideas to present them in a better way. The backend part (tokenization and views) wouldn't be possible without the help of the mentors. Thank you so much @philippe, @tdruez, and @Ayan!

- A more detail view of licenses per match
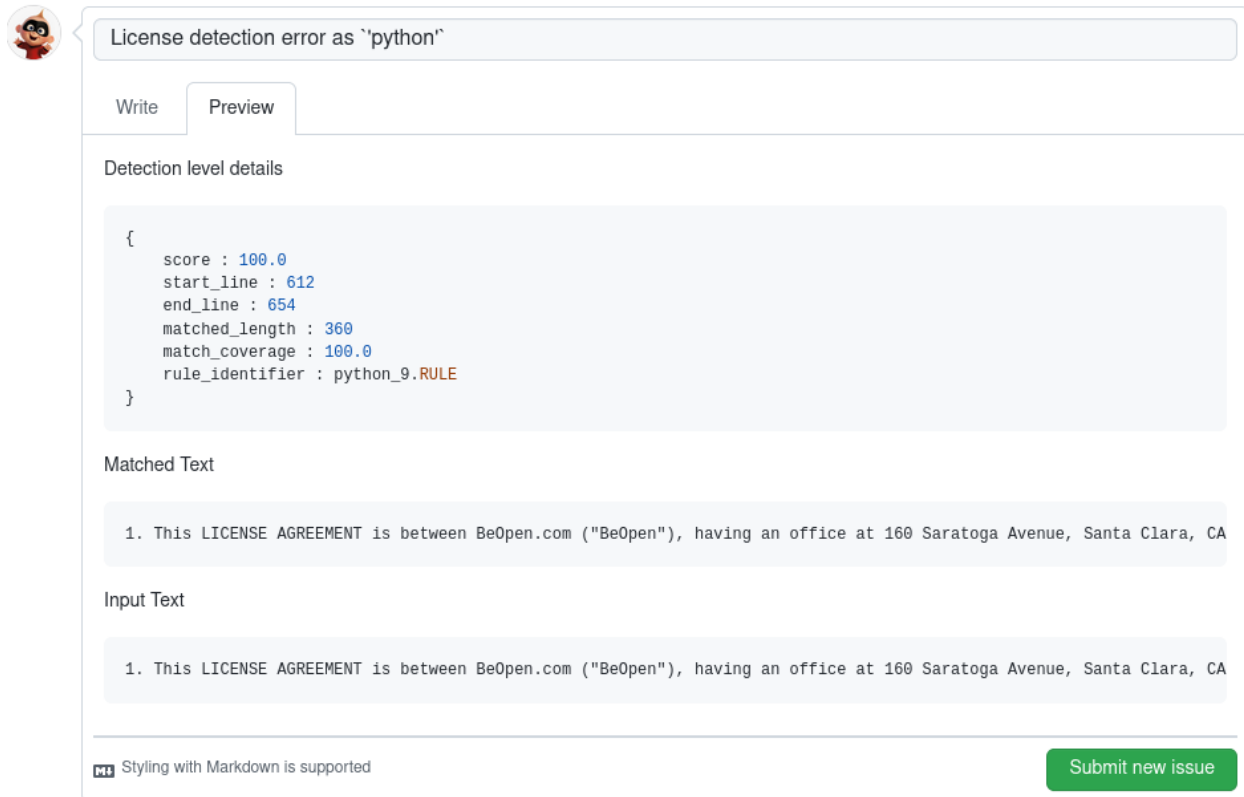
Now the user can see what has been detected, right? What if the user concludes there is something wrong with the match?

Well, to solve it, the user has to create a new issue on scancode-toolkit's github page and upload a few details so the developers can understand what the issue is actually. For that, the user has to provide the *Input Text*, *Matched Text*, and *Detected Values*. Luckily, there is a report functionality button provided inside the application which on click does all the work for the end user and redirects the user to scancode-toolkit's issues page on github to save some time.

License detection error as ``python``

Write   Preview

Detection level details

```
{
    score : 100.0
    start_line : 612
    end_line : 654
    matched_length : 360
    match_coverage : 100.0
    rule_identifier : python_9.RULE
}
```

Matched Text

```
1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA
```

Input Text

```
1. This LICENSE AGREEMENT is between BeOpen.com ("BeOpen"), having an office at 160 Saratoga Avenue, Santa Clara, CA
```

Styling with Markdown is supported   Submit new issue

Post GSoC

I am interested in continuing the development process of the Scancode.io. There are still things needed to be implemented which are necessary for having an improved and stable version of *scantext* inside scancode.io.

Links

- Project Idea
- GSoC Proposal
- Source Code
- Merged Commits

Acknowledgements

I had an amazing time working on this project this summer. Thanks, for selecting me for this project. I've got an opportunity to learn various things about software licensing and I was also able to learn various tools and got deep into Django and Django-REST-framework.

Thanks for guiding me.

Tabs are Evil, Space Rocks!!!

Refactor Scancode Workbench to React-Typescript

By, Omkar Phansopkar
Project: Scancode-Workbench
Organization - AboutCode

Overview

ScanCode Workbench provides an advanced visual UI to help you quickly evaluate license and other notices identified by ScanCode. ScanCode detects licenses, copyrights, packages and other interesting information in your code that can be visualised efficiently using workbench.

The GSoC project's goal was to refactor ScanCode Workbench to a React + Typescript implementation and improve various sections of the application including Table view, file uploads, data sync across components, etc. Plus, the workbench wasn't updated for a long time, to support the latest scans from scancode-toolkit, hence the schema and UI had to be changed accordingly.

Goals achieved

- Refactor Workbench from vanilla javascript to React + Typescript implementation

- Update Workbench to support the latest scancode-toolkit output format. (currently supports outputs from scancode-toolkit v31.1.1)

- Improve User experience in Tableview, Home page, etc

- Add convenience features like Drop files, History records, etc.

- Add new sections for Scan Info, packages-dependencies info, etc

- Automated releases using github actions (except for apple-silicon builds)

Quick look

Links

- Download ScanCode Workbench beta
- Download latest ScanCode Workbench
- GSoC Project Details
- Proposal
- ScanCode Workbench
- ScanCode Toolkit

Post GSoC - Future Plans

I wish to carry on with the development of Scancode-workbench and implement the ideas suggested by my mentors. Workbench isn't yet utilising all the scan data it imports, I wish to propose best ways to express this data. Moreover, with new release formats of scancode-toolkit, workbench must adapt and update its schema & UI at pace with scancode-toolkit.

Closing Thoughts

It was a great experience contributing to Workbench, Aboutcode team provided me with the freedom to implement UI improvements and suggested a lot of improvements throughout the journey. I learned a lot of things from working on an electron app to creating automated GitHub release scripts.

Heartiest thanks to @steven-esser @AyanSinhaMahapatra @TG1999 @pombredanne and all the About code members for your constant feedback and support over the course of the project.

You can connect with me here
https://github.com/OmkarPh
https://www.linkedin.com/in/omkarphansopkar

To the reader, Have a look at aboutcode's amazing projects. Surely you'll find something to contribute that suits your preference

See ya

VulnTotal: Tool for cross-validating vulnerability

Organization - AboutCode

Keshav Priyadarshi
GitHub: keshav-space
LinkedIn: @keshav-space
Project: VulnTotal
Proposal: Link

Overview

VulnTotal `cross-validates` the vulnerability coverage of publicly available vulnerability check tools and databases. It's inspired by the VirusTotal multi-scanner virus scanning service. There are scenarios where a package is reported as vulnerable by some tools or databases but not by others, VulnTotal helps in detection such anomaly. We can gradually work with these tool providers to keep each other apprised about newly discovered vulnerabilities and anomaly, making FOSS more secure.

Sneak Peek

Fig. 1: VulnTotal takes PURL as an argument and returns vulnerability data from various data sources. By default, vulnerability data is grouped by CVE.

Note: A PURL is a URL string used to identify and locate a software package in a mostly universal and uniform way across programming languages, package managers, packaging conventions, tools, APIs and databases. more on PURL

VulnTotal Development - Walkthrough

Initial Configuration

The initial PR and commits outlined the core structure and implemented `VendorData` and `DataSource` inside `validator.py`.

VendorData is dataclass that encapsulates `aliases`, `affected_versions` and `fixed_versions` for a vulnerability.

DataSource outlines core methords such as `datasource_advisory` and `supported_ecosystem` to be implemented by subclass.

Below is the tree view of VulnTotal for better understanding

```
vulntotal
├── validator.py
├── vulntotal_cli.py
├── vulntotal_utils.py
├── datasources
│      ├── __init__.py
│      ├── deps.py
│      ├── github.py
│      ├── gitlab.py
│      ├── oss.py
│      ├── osv.py
│      ├── snyk.py
│      └── vulnerablecode.py
└── tests
       ├── test_deps.py
       ├── test_github.py
       ├── test_oss.py
       ├── test_osv.py
       ├── test_snyk.py
```

<div align="right">(continues on next page)</div>

```
        ├── test_vulnerablecode.py
        └── test_data
                ├── deps/
                ├── github/
                ├── oss_index/
                ├── osv/
                ├── snyk/
                └── vulnerablecode/
```

PR and commits related to initial configuration

- [nexB/vulnerablecode#777](#)

- [nexB/vulnerablecode#2176cb11](#)

- [nexB/vulnerablecode#922859f3](#)

- [nexB/vulnerablecode#78dd5ae7](#)

Adding DataSource

The initial config made adding datasource fairly smooth. AnyNewDataSource just needed to inherit `DataSource` and implement `datasource_advisory` and `supported_ecosystem`

datasource_advisory is core method that takes PURL as an arguments and yields `VendorData`

supported_ecosystem should return a dictionary that maps PURL equivalent of ecosystem (aka purl.type) to Data-Source equivalent ecosystem.

Currently Supported DataSource

1. Open Source Vulnerability ⟨osv.dev⟩

OSV provides API end-point for querying package vulnerability. Unfortunately NuGet package names aren't case normalized by OSV. So the OSVDataSource employs NuGet SearchQueryService for discovering the valid case-sensitive package name and then uses that to query OSV. For more on this issue see [nexB/vulnerablecode/#800](#)

Related PR: [nexB/vulnerablecode#788](#)

2. Open Source Insights ⟨deps.dev⟩

Writing datasource for deps was quite uneventful. Deps doesn't provide any documented API except for GCP BigQuery, but it does have obfuscated API and DepsDataSource makes use of that.

Related PR: [nexB/vulnerablecode#789](#)

3. GitHub Advisory Database

GitHub provide GraphQL end-point for querying package vulnerability, but it comes with a caveat that one can't query a specific version of a particular package. It dumps vulnerability related to all the versions of a particular package. For this vulntotal_utils implements a specialized method `github_constraints_satisfied` to filters out vulnerabilities for specific version.

Related PR: nexB/vulnerablecode#804

4. Sonatype OSS Index

OSSIndexDataSource makes use of oss-index API. OSS-Index only provides CVE's related particular package version and makes no mention of either the affected package versions or fixed package version.

Related PR: nexB/vulnerablecode#829

5. VulnerableCode Advisory Database

VulnerableCodeDataSource currently make use of local VulnerableCode instance, but soon will be migrated to global instance.

Related PR: nexB/vulnerablecode#832

6. Snyk Vulnerability Database

Snyk comes with no API whatsoever, so had to restore to web scrapping using BeautifulSoup. A specialized method `snky_constraints_satisfied` was implemented just filter out vulnerabilities for specific version. Among all the datasources currently available, Snyk is the only one that keeps track of malicious packages.

Related PR: nexB/vulnerablecode#842

7. GitLab Gemnasium Advisory Database

Again, GitLab comes with no API, so GitlabDataSource is designed to directly fetch package vulnerability data from GitLab gemnasium repository. For case-sensitive package name, GitLab GraphQL end-point is used to get the exact case-sensitive package name. A similar method `gitlab_constraints_satisfied` is implemented to filter out vulnerabilities for specific version.

Related PR: nexB/vulnerablecode#883

Automatic Datasourse Registery

All new Datasource must be added to `DATASOURCE_REGISTERY` to make it available for use. Fortunately `__init__.py` is configured to take care of this, as soon as a new and valid datasource file is added inside datasources directory it automatically gets registered and vice versa.

Related PR: nexB/vulnerablecode#901

Command-line Interface

VulnTotal CLI takes PURL as an argument and returns vulnerability data from various data sources. By default, vulnerability data is grouped by CVE. It also supports JSON and YAML data dump. Since most datasources are Network I/O intensive, so by default CLI makes use of ThreadPoolExecutor for better efficiency.

Related PR: nexB/vulnerablecode#801

Tip:

CLI comes with lots of hidden features that are specially useful while debugging a datasource.
Look inside `vulntotal_cli.py` to discover them all.

Pre GSoC

- Test sorting of all the OpenSSL versions ever released. nexB/univers#61
- Migrate OpenSSL importer to importer-improver model. nexB/vulnerablecode#690
- Correct notes for cvssv3.1_qr. nexB/vulnerablecode#599
- Add from_versions in VersionRange. nexB/univers#55
- Add OpenSSL support in univers. nexB/univers#42
- Fix for NpmVersionRange.from_native and README. nexB/univers#34
- Add black code-style test for skeleton. nexB/skeleton#56

Post GSoC - Future Plans & Suggestions

- Support query using aliases. nexB/vulnerablecode/#824
- Adding more DataSource like mend.io. nexB/vulnerablecode/#835
- Support for API and Web UI.
- Cluster analysis of advisory fetched from different DataSources. nexB/vulnerablecode#822
- Handle forever vulnerable packages in VulnerableCode nexB/vulnerablecode#855

Closing Thoughts

Thoroughly enjoyed working on this project. Weekly calls were greatly helpful and thanks to Philippe, Hritik, Tushar, Shivam for the thoughtful inputs. Learned a lot about various interesting projects and what it takes to tame some of the real world problems. Greatly enhanced my ability to conduct myself in an open source world. All in all it's been a remarkable journey.

Add more data sources and mine the graph to find correlations between vulnerabilities

By, Ziad Hany
Project: VulnerableCode
Organization - AboutCode

Project Idea

There is a large number of pending tickets for data sources. We want to search for more vulnerability data sources and consume them . So I will resolve this issues :

- Add OSV importer #608 (add support for more data source like OSS-Fuzz ,…)
- Add support for CWE #651
- Collect UVI database #588
- Collect CMU CERT #588
- Add Fireeye disclosures #487

Tasks Accomplished

- add a OSV importer
- Refactor Git-importer using fetchcode
- Import data from Fireeye
- Add support for CWE
- Import data from GSD
- Migrate ruby to new importers

Links

- Official GSoC project page
- GSoC Proposal
- Project Idea

Post GSoC - Future Plans

- add support for Cargo version and Version range
- add support for Git Version and Version range , …

I had a wonderful time during the Gsoc, I learned a lot of things during this time and I would like to thanks @TG1999 @pombredanne I learnt a lot from you guys

You can connect with me here
https://github.com/ziadhany
https://www.linkedin.com/in/ziadhany

2.2.2 GSoC 2021

Virtual Codebase support

Project: Virtual Codebase support in DeltaCode

Pratik Dey <pratikrocks.dey11@gmail.com>

Project Overview

The goal of this proposal is to refactor DeltaCode to use Scancode-Toolkit's Virtual Codebase class. This refactoring will allow DeltaCode to be a library as opposed to only be used as a CLI tool, moreover, this refactor will allow DeltaCode to determine deltas much more effectively in the form of BFS tree scan of the two tree structures unlike indexing the entire codebase.

Main Objectives of the project

- Migrate to using VirtualCodebase from the latest scancode.
- Create DeltaCode documentation on Read The Docs.
- Provide the support for fingerprint plugin for Virtual Codebase.
- Provide the Support for enabling Virtual Codebase to scan files having full root paths as their location.

The Project

- Virtual Codebase Integration with Deltacode
- Removing redundant File and License Objects
- Provided options in deltacode scans
- Added Docker Script for Dockerizing the Deltacode Application and make it platform-independent.
- Add Read the Docs Support to Deltacode.

I have completed all the tasks that were in the scope of this GSoC project.

Pull Requests

- https://github.com/nexB/deltacode/pull/167 [Merged]
- https://github.com/nexB/deltacode/pull/176 [Open]
- https://github.com/nexB/deltacode/pull/171 [Open]
- https://github.com/nexB/deltacode/pull/178 [Open]
- https://github.com/nexB/deltacode/pull/168 [Open]

Links

- Project Details
- Proposal
- ScanCode Toolkit
- DeltaCode

I've had a wonderful time during these three months and have learned plenty of things. I would really like to thank @pombredanne, @majurg, and @JonoYang for their constant support throughout the journey. From good job claps to nit-picky constructive code-reviews, I enjoyed every bit of this GSoC project.

I had a wonderful time during the GSOC, I learned a lot of things during this time. I really enjoyed this project. I would really like to thank my mentors @pombredanne, @majurg, and @TG1999, and all other About code members who constantly supported me throughout this project.

Unknown License Detection and License References

Organisation - AboutCode

Akanksha Garg <akanksha.garg2k@gmail.com>

GITHUB

Project: Detect Unknown Licenses and Indirect License References in Scancode

ScanCode-toolkit

Project Link

Proposal

Description

The main motive of this project was to improve license detection of unknown licenses and follow references to indirect license references in Scancode-TK

Improvement in the License Data Model Definition

Unknown Licenses are the ones which are matched to a license rule tagged with 'unknown' license key. Since these are some of the 'special' licenses , reporting them with special attributes will provide more clarification. Now unknown licenses are tagged with a new flag **"is_unknown"** to identify them beyond just the naming convention of having "unknown" as part of their name.

Rules that match at least one unknown license have a flag **"has_unknown"** set in the returned match results.

nexB/scancode-toolkit#2548

Reporting known and Unknown licenses separately

We considered having a separate section for of scan results to report 'unknown licenses' separately and not mixed with main license detection results. But after implementing a separate section for unknown ones ,it doesn't seem to be good idea to have currently.

nexB/scancode-toolkit#2578

Follow License References to another file

Some license references such as "see license in file LICENSE.txt" e.g. mentions to look for license details in another file are reported as unknown license references and we could instead follow the referenced file to find what was detected there. The approach was to use already contained attribute `refrenced_filenames` in license RULE data files. Since this was a `process_codebase` step in scan plugin , it was needed that our API function should return `refrenced_filenames` to keep track of these files corresponding to licenses detected. This was tracked in -

nexB/scancode-toolkit#2632

The `process_codebase` step is tracked in -

nexB/scancode-toolkit#2616

Improve license detection of Unknown Licenses

The approach was to use index of n-grams for detecting unknowns besides having our actual detection of "unknown" license rules. Firstly matches were filtered after running our normal procedure of license detection and the remaining spans are run through a automaton index containing n-grams from all regular license texts and rules. This is tracked in -

nexB/scancode-toolkit#2592

Addition of some new Licenses

There were some licenses that were not present in Scancode-toolkit as for now. They have been added now.

nexB/scancode-toolkit#2625

Pre-GSoC

Contributions

- nexB/scancode-toolkit#2423
- nexB/scancode-toolkit#2473
- nexB/scancode-toolkit#2464
- nexB/scancode-toolkit#2381

I've had a wonderful summer during these 10 weeks journey and have learned plenty of things. I am thankful to Google and Aboutcode for giving me this opportunity to work with such an amazing community. I am fortunate to have mentors Philippe Ombredanne and Ayan Sinha Mahapatra who helped me a lot throughout my GSoC project and provided constant support.

Vulnerablecode refactor and add Importers

Organization - AboutCode

Hritik Vijay
Project: VulnerableCode

Overview

VulnerableCode is a decentralized python program to collect data about open source software vulnerabilities across the internet. My proposal for this year's Google Summer of Code involved improving the import speed, refactoring existing code, finding points for overall improvement and adding importers.

Detailed Report

Improve Import Time

Profiling showed that a lot of time was being wasted during auto commits undertaken by django. Wraping the importer in an atomic block avoids lots of database commits and shows huge performance improvement. This simple change allows for much faster import times while not drastically changing the code structure:

```
Alpine: 202.7s -> 50.9s
Archlinux 2116.6s -> 107.8s
Gentoo 3176.3s -> 225.8s
```

Yielding an average of 93% reduction in time (14x faster)

More: https://github.com/nexB/vulnerablecode/pull/478

Speed up upstream tests

VulnerableCode performs upstream tests for all the importers to make sure that any change change in upstream data structure is easily spotted. This allows us to have a look at failing importers without actually deploying the application.

Earlier, all of the importers were run one by one in order to verify that they are intact. While this being the obvious and the full proof way to detect any anomalies in the imported data schema, it did not work because the time required to run all the importers much exceeded 6 hours - which is the maximum time allowed for GitHub actions to run. With this PR, the updated_advisories method of each importer is expected to create at least one Advisory object. If it does so, the importer is marked working. While this is not full proof, it stays much below the allowed resource usage cap. In the end, this is a trade off between resource usage and data accuracy. This brings major performance improvement during the test.

Before: ~6hrs, now ~9 minutes
More: https://github.com/nexB/vulnerablecode/pull/490

Improve Docker Configuration

The preferred mode of deployment for VulnerableCode is deploying using Docker images. Docker configuration existing earlier was very insecure and rudimentary. I took the inspiration for a uniform Docker configuration from the ScanCodeIO project and provided with detailed documentation for installation using a docker image. The current configuration makes use of files like `docker.env` to supply container's environment and `.dockerignore` to skip over any unnecessary files for deployment.

More:

https://github.com/nexB/vulnerablecode/pull/497

https://github.com/nexB/vulnerablecode/pull/521

Add Makefile

Makefile usage is prevalent in sister projects like ScanCodeIO. It gives VulnerableCode a consistent behavior and provides a very friendly interface for invocations. This also avoids security risks like having a default django `SECRET_KEY` as it can be easily generated by a make target. I added a Makefile which has a similar usage as that of ScanCodeIO, replaced all the CI tests to use make, updated the relevant part of the documentation and updated settings to reject insecure deployments.

More:

https://github.com/nexB/vulnerablecode/pull/497

https://github.com/nexB/vulnerablecode/pull/523

Use svn to collects tags in GitHubTagsAPI

Surprisingly, GitHub allows svn requests to repositories. Now we can have all the tags with a single request. This is much more efficient and gentle to the APIs. This was as issue since the importers based on GithubDataSource were failing because of being rate limited by GitHub.

Philippe, thank you so much for the suggestion

More: https://github.com/nexB/vulnerablecode/pull/508

Separate import and improve operations - WIP

This introduces a new concept of `improver`. Earlier, data fetching and improvement were done as one single process by `importer`. This meant that importers were convoluted and not very modular. The concept of `improver` comes from the idea that an `importer` should only do one thing - import. Any further improvement on the data is delegated to the improvers. This allows for us to have multiple ways of improvement with certain confidence on the improved data making the import and improve operations modular and simpler to work with. As a bonus, writing importers will be very easy and welcome more contributors to the project. As of writing this report, this remains a work in progress which will be finished very soon.

More: https://github.com/nexB/vulnerablecode/pull/525

Others

- helper: split_markdown_front_matter: https://github.com/nexB/vulnerablecode/pull/443

- Dump yaml in favor of saneyaml https://github.com/nexB/vulnerablecode/pull/452

- Refactor package_managers https://github.com/nexB/vulnerablecode/pull/495/commits

- Importers bugfix https://github.com/nexB/vulnerablecode/pull/505

Pre GSoC

I started to like VulnerableCode as soon as I laid eyes on the project. While exploring the codebase, I realized that there is a lot of room for improvement. Thus I looked for simple improvements and bugs to fix in the early stage, which were:

- Correct API docs path and fix pytest invocation
- Explicity provide lxml parser to beautifulsoup
- Make sure vulnerability id is_cve or is_vulcoid
- Fix istio importer (cleared a huge confusion about the codebase)
- Add me to AUTHORS (Should've done this a lot earlier)
- Add unspecified scoring system
- Fix redhat import failure (This one took a *lot* of effort to pinpoint)
- expose find_all_cve helper

Post GSoC - Future Plans and what's left

I wish to carry on with the development of VulnerableCode and implement the ideas suggested by my mentors. This will require a lot of effort to bring VulnerableCode to a stable point. I hope to see VulnerableCode integrated into the ScanCode toolkit happen in a near future.

Further, if possible, I would like VulnerableCode to interact with other great open source tools like *Eclipse Steady* and *Prospector*. VulnerableCode, currently, works statically to collect all the vulnerabilities from different data sources, meanwhile there have been some developments with the Prospector project of Eclipse Steady. The project aims to scan fix-commits of the git repository in order to find out if the vulnerable part of a library was actually used in a project. It is not always the case that if a library is vulnerable then all the projects building upon it would be vulnerable too. It is crucial to identify if it is worth updating the library in use and dealing with the breaking changes. *Prospectus* is undergoing improvements in order to be released as a usable public tool. *Project KB* (Under Eclipse Steady) is also working on a "tool support for mining repositories and databases of advisories to establish the (missing) link between vulnerabilities (as described in natural language in the advisories) and the corresponding fix-commits". When these projects are ready for public use I would like to add them to VulnerableCode as a modules. I hope this will benefit both the projects and the downstream.

After everything mentioned above, writing importers and improvers is something that is still left. In my opinion, this needs to be addressed after having a stable structure for VulnerableCode.

Closing Thoughts

I really enjoyed working on the project. There were ups and downs when I met some weird bugs but every one of them taught me something new about Python, Django and programming in general. The best part of working with my amazing mentors - Philippe and Shivam - were the weekly meets where we would together try to figure out how to proceed with the development. I learned something new with every call and interaction we had. Thank you so much my mentors for providing a very smooth experience and Google for showing me the guiding light for participation.

To the reader, I would really like you to read this before Philippe asks you to ;)

2.2.3 GSoC 2020

Support Python 3.x Part 2

Goals

- Remove legacy Python code

- Support latest Python 3.x

- Improve 3rd party package provisioning

ScanCode has been ported on Python 3 but still supports Python 2.7. The goal of this project is to remove all the code that provides Python 2-specific support.

In addition, the way we deal with 3rd party packages needs to be refactored: today we embed a copy of all our third-party dependencies as pre-built wheels in the ScanCode repository. This creates some problems:

1. We supported Python 2 and now support Python 3.6. Each version may need a pre-built wheel if there is native code. With 3.7, 3.8 and beyond this means creating and storing in the ScanCode git repo 4 variants for each dependency.

2. We support Windows, Linux, macOS and possibly FreeBSD each with some possible architecture variants. For each of these we are storing a variant for each of the Python versions above.

This creates way too many wheels that end up making ScanCode checkouts and tarballs too big. We need to define a new improved way to handle third-party dependencies including:

- Make the build automation more prominent

- Ensure secured pinning of wheel versions and actual files (e.g. using checksums)

- Do not bundle these wheels anymore.

For more info see Disk quota and Limitations.

Basically it is an extension of previous GSoC 2019 Project: Port scancode-toolkit to Python 3

Assigned to

Abhishek Kumar: https://github.com/Abhishek-Dev09

Level

Advanced

Tech

Python, build scripts, packaging

URLS

https://github.com/nexB/scancode-toolkit/issues/295

Project Board

https://github.com/nexB/scancode-toolkit/projects/8

Mentors

@majurg https://github.com/majurg @pombredanne https://github.com/pombredanne

Commits

List of commits(merged): https://github.com/nexB/scancode-toolkit/commits?author=Abhishek-Dev09 https://github.com/nexB/scancode-toolkit/pull/2184 (open) https://github.com/nexB/scancode-toolkit/pull/2167 (open)

2.2.4 GSoC 2019

Approximately similar file detection

Project: Approximately similar file detection in DeltaCode

Arnav Mandal <arnav.mandal1234@gmail.com>

Project Overview

DeltaCode is a tool to compare and report scan differences. It takes JSON files as an input which is the output of ScanCode-toolkit as well. When comparing files, it only uses the exact comparison. By exact comparison, I mean it compares the hash value of the files. The output of DeltaCode is a JSON/CSV file which includes the details of the scan such as delta score, delta count, etc. The goal of this project is to improve the usefulness of the delta by also finding files that are mostly the same (e.g. quasi or near duplicates) vs. files that are completely different. After this project, DeltaCode would be able to detect similar files in a directory approximately.

Requirements of the project

- Provided two files using ScanCode-toolkit, the new near-duplicate detection should return the distance between the two files.

- The code should be seamlessly integrated with ScanCode-toolkit. It should be highly configurable by the maintainers.

- The strictness of near-duplicates should be noted and adjusted by a threshold variable.

The Project

- Addition of new fingerprint plugin in the ScanCode Toolkit.

- Implementation and integration of the fingerprint generation algorithm in the ScanCode Toolkit codebase.

- Implementation of distance finding algorithm between the files and process them further in the DeltaCode codebase.

- Integration of fingerprint field in the JSON file to compare the deltas and provide them with appropriate scores.

- Make changes to old unit tests and addition of new unit tests in ScanCode Toolkit as well as DeltaCode.

I have completed all the tasks that were in the scope of this GSoC project.

Pull Requests

- https://github.com/nexB/scancode-toolkit/pull/1576 [Closed] (something went wrong while rebasing)

- https://github.com/nexB/scancode-toolkit/pull/1651 [Merged]

- https://github.com/nexB/deltacode/pull/128 [Merged]

Links

- Project Details
- Proposal
- ScanCode Toolkit
- DeltaCode

I've had a wonderful time during these three months and have learned plenty of things. I would really like to thank @pombredanne, @steven-esser, and @JonoYang for their constant support throughout the journey. From good job claps to nit-picky constructive code-reviews, I enjoyed every bit of this GSoC project.

Port scancode-toolkit to Python 3

Project: scancode-toolkit to Python 3

Owner: Abhishek Kumar

Mentor: Philippe Ombredanne

Overview

Problem: Since Python 2.7 will retire in few months and will not be maintained any longer.

Solution: Scancode needs to be ported to python 3 and all test suites must pass on both version of Python. The main difference that makes Python 3 better than Python 2.x is that the support for unicode is greatly improved in Python 3. This will also be useful for scancode as scancode has users in more than 100 languages and it's easy to translate strings from unicode to other languages.

Objective: To make scancode-toolkit installable on on Python 3.6 and higher, as presently it installs with Python 2.7 only.

Implementation

- It was started in development mode(editable mode) and then it was moved to work in virtual environments.

- I have worked module by module according to the order of hierarchy of modules. For example :All module is dependent on commoncode, so it must be ported first. In this way we have created the Porting order:

    1. commoncode

    2. plugincode

    3. typecode

    4. extractcode

    5. textcode

    6. scancode basics (some tests are integration tests and will have to wait to be ported)

    7. formattedcode, starting with JSON (some tests are integration tests and will have to wait to be ported)

    8. cluecode

    9. licensedcode

    10. packagedcode (depends on licensecode)

    11. summarycode

    12. fixup the remaining bits and tests

After porting each module, I have marked these modules as ported `scanpy3` with help of **conffest** plugin (created by @pombredanne). **Conffest** plugin is heart of this project. Without this, it was very difficult to do. Dependencies was fixed at the time of porting the module where it was used.

Challenging part of Project

It is very difficult to deal with paths on different operating systems.The issue is around macOS/Windows/Linux. The first two OS handle unicode paths comfortably on Python 2 and 3 but not completely on macOS Mojave because its filesystem encoding is APFS. Linux paths are bytes and os.listdir is broken on Python 2. As a result you can only sanely handle Linux paths as bytes on Python 2. But on Python 3 path seems to be corrected as `unicode` on Linux.

For more details visit here :

- https://vstinner.github.io/painful-history-python-filesystem-encoding.html
- jaraco/path.py#130

We came with various Solution:

- To use pathlib which generally handle paths correctly across platforms. And for backports we use pathlib 2. But this solution also fails because pathlib 2 does not work as expected wrt unicode vs bytes. And os.listdir also doesn't work properly.
- To use path.py which handles the paths across all the platforms even on macOS Mojave .
- Use `bytes` on linux and python 3 and `unicode` everywhere.

We choose the third solution because it is most fundamental and simple and easy to use.

Project was tracked in this ticket nexB/scancode-toolkit#295

Project link : Port Scancode to Python 3

My contribution : List of Commits

Note : Please give your feedback here

Outcome

Now we have liftoff on Python 3 . We are able to run basic scans without errors on develop branch. You check it by running `scancode -clipeu samples/ --json-pp - -n4` .

At last I would like to thanks my Mentor **@pombredanne** aka Philippe Ombredanne . He has helped lot in completing this project. He is very supportive and responsive. I have learned a lot from him. By his encouragement and motivation, I am very improving day by day, building and developing my skills. I have completed all the tasks that were in the scope of this GSoC project.

2.2.5 GSoC 2017

Plugin architecture

Project: Plugin architecture for ScanCode

Yash D. Saraf yashdsaraf@gmail.com

This project's purpose was to create a decoupled plugin architecture for ScanCode such that it can handle plugins at different stages of a scan and can be coupled at runtime. These stages were,

1. Format :

In this stage, the plugins are supposed to run **after** the scanning is done and `post-scan` plugins are called. These plugins could be used for:

- **converting the scanned output to the given format (say csv, json, etc.)**

HOWTO

Here, a plugin needs to add an entry in the `scancode_output_writers` entry point in the following format : `'<format> = <module>:<function>'`.

- `<format>` is the format name which will be used as the command line option name (e.g `csv` or `json`).
- `<module>` is a python module which implements the `output` hook specification.
- `<function>` is the function to which the scan output will be passed if this plugin is called.

The `<format>` name will be automatically added to the `--format` command line option and (if called) the scanned data will be passed to the plugin.

2. Post-scan :

In this stage, the plugins are supposed to run **after** the scanning is done. Some uses for these plugins were:

- **summarization of scan outputs**

    e.g A post-scan plugin for marking `is_source` to true for directories with ~90% of source files.

- **simplification of scan outputs**

    e.g The `--only-findings` option to return files or directories with findings for the requested scans. Files and directories without findings are omitted (not considering basic file information as findings)).

This option already existed, I just ported it to a post-scan plugin.

HOWTO

Here, a plugin needs to add an entry in the `scancode_post_scan` entry point in the following format `'<name> = <module>:<function>'`

- `<name>` is the command line option name (e.g **only-findings**).
- `<module>` is a python module which implements the `post_scan` hook specification.
- `<function>` is the function to which the scanned files will be passed if this plugin is called

The command line option for this plugin will be automatically created using the `<function>` 's doctring as its help text and (if called) the scanned files will be passed to the plugin.

3. Pre-scan :

In this stage, the plugins are supposed to run **before** the scan starts. So the potential uses for these types of plugins were to:

- **ignore files based on a given pattern (glob)**
- **ignore files based on their info i.e size, type etc.**
- **extract archives before scanning**

HOWTO

Here, a plugin needs to add an entry in the `scancode_pre_scan` entry point in the following format : `'<name> = <module>:<class>'`

- `<name>` is the command line option name (e.g **ignore**).

- `<module>` is a python module which implements the `pre_scan` hook specification.

- `<class>` is the class which is instantiated and its appropriate method is invoked if this plugin is called. This needs to extend the `plugincode.pre_scan.PreScanPlugin` class.

The command line option for this plugin will be automatically created using the `<class>` 's doctring as its help text. Since there isn't a single spot where `pre-scan` plugins can be plugged in, more methods to `PreScanPlugin` class can be added which can represent different hooks, say to add or delete a scan there might be a method called `process_scan`.

If a plugin's option is passed by the user, then the `<class>` is instantiated with the user input and its appropriate aforementioned methods are called.

4. Scan (proper):

In this stage, the plugins are supposed to run **before** the scan starts and **after** the `pre-scan` plugins are called. These plugins would have been used for

- **adding or deleting scans**
- **adding dependency scans (whose data could be used in other scans)**

No development has been done for this stage, but it will be quite similar to `pre-scan`.

5. Other work:

Group cli options in cli help

Here, the goal was to add command line options to pre-defined groups such that they are displayed in their respective groups when `scancode -h` or `scancode --help` is called. This helped to better visually represent the command line options and determine more easily what context they belong to.

Add a Resource class to hold all scanned info * `Ongoing` *

Here, the goal was to create a `Resource` class, such that it holds all the scanned data for a resource (i.e a file or a directory). This class would go on to eventually encapsulate the caching logic entirely. For now, it just holds the `info` and `path` of a resource.

6. What's left?

- Pre-scan plugin for archive extractions
- Scan (proper) plugins
- More complex post-scan plugins
- Support plugins written in languages other than python

Additionally, all my commits can be found here.

2.2.6 GSoC project ideas pages

Google Summer of Code 2019

AboutCode is participating in the Google Summer of Code in 2019 as a mentoring org. This page contain all the information for students and anyone else interested in helping.

AboutCode is a family of FOSS projects to uncover data ... about software code:

- where does the code come from? which software package?

- what is its license? copyright?

- is the code secure, maintained, well coded?

All these are questions that are important to answer: there are million of free and open source software components available on the web for reuse.

Knowing where a software package comes from, what is its license and if it is vulnerable and what's its licensing should be a problem of the past such that everyone can safely consume more free and open source software.

Join us to make it so!

Our tools are used to help detect and report the origin and license of source code, packages and binaries as well as discover software and package dependencies, and in the future track security vulnerabilities, bugs and other important software package attributes. This is a suite of command line tools, web-based and API servers and desktop applications.

Table of Contents

AboutCode projects are. . .

- ScanCode Toolkit is a popular command line tool to scan code for licenses, copyrights and packages, used by many organizations and FOSS projects, small and large.

- Scancode Workbench (formerly AboutCode Manager) is a JavaScript, Electron-based desktop application to review scan results and document your origin and license conclusions.

- AboutCode Toolkit is a command line tool to document and inventory known packages and licenses and generate attribution docs, typically using the results of analyzed and reviewed scans.

- TraceCode Toolkit is a command line tool to find which source code file is used to create a compiled binary and trace and graph builds.

- DeltaCode is a command line tool to compare scans and determine if and where there are material differences that affect licensing.

- ConAn : a command line tool to analyze the code in Docker and container images

- VulnerableCode : an emerging server-side application to collect and track known package vulnerabilities.

- license-expression : a library to parse, analyze, simplify and render boolean license expression (such as SPDX)

We also work closely, contribute and co-started several other orgs and projects:

- Package URL which is an emerging standard to reference software packages of all types with simple, readable and concise URLs.

- SPDX aka. Software Package Data Exchange, a spec to document the origin and licensing of packages.

- ClearlyDefined to review and help FOSS projects improve their licensing and documentation clarity.

Contact

Join the chat online or by IRC at https://gitter.im/aboutcode-org/discuss Introduce yourself and start the discussion!

For personal issues, you can contact the primary org admin directly: @pombredanne and pombredanne@gmail.com

Please ask questions the smart way: http://www.catb.org/~esr/faqs/smart-questions.html

Technology

Discovering the origin of code is a vast topic. We primarily use Python for this and some C/C++ (and eventually some Rust and Go) for performance sensitive code and Electron/JavaScript for GUI.

Our domain includes text analysis and processing (for instance for copyrights and licenses detection), parsing (for package manifest formats), binary analysis (to detect the origin and license of binaries, which source code they come from, etc.) as well as web based tools and APIs (to expose the tools and libraries as web services) and low-level data structures for efficient matching (such as Aho- Corasick and other automata).

Skills

Incoming students will need the following skills:

- Intermediate to strong Python programming. For some projects, strong C/C++ and/or Rust is needed too.
- Familiarity with git as a version control system
- Ability to set up your own development environment
- An interest in FOSS licensing and software code and origin analysis

We are happy to help you get up to speed, but the more you are able to demonstrate ability and skills in advance, the more likely we are to choose your application!

About your project application

We expect your application to be in the range of 1000 words. Anything less than that will probably not contain enough information for us to determine whether you are the right person for the job. Your proposal should contain at least the following information, plus anything you think is relevant:

- Your name
- Title of your proposal
- Abstract of your proposal
- Detailed description of your idea including explanation on why is it innovative and what it will contribute to the project
    - hint: explain your data structures and you planned main processing flows in details.
- Description of previous work, existing solutions (links to prototypes, bibliography are more than welcome)
- Mention the details of your academic studies, any previous work, internships
- Relevant skills that will help you to achieve the goal (programming languages, frameworks)?
- Any previous open-source projects (or even previous GSoC) you have contributed to and links.
- Do you plan to have any other commitments during GSoC that may affect your work? Any vacations/holidays? Will you be available full time to work on your project? (Hint: do not bother applying if this is not a serious full time commitment during the GSoC time frame)

Join the chat online or by IRC at https://gitter.im/aboutcode-org/discuss introduce yourself and start the discussion!

The best way to demonstrate your capability would be to submit a small patch ahead of the project selection for an existing issue or a new issue.

We will **always** consider and prefer a project submissions where you have submitted a patch over any other submission without a patch.

You can pick any project idea from the list below. If you have other ideas that are not in this list, contact the team first to make sure it makes sense.

Our Project ideas

Here is a list of candidate project ideas for your consideration. Your own ideas are welcomed too! Please chat about them to increase your chances of success!

ScanCode ideas

Improve Copyright detection accuracy and speed in ScanCode

Copyright detection is reasonably good by the slowest scanner in ScanCode. It is based on NLTK part of speech (PoS) tagging and a copyright grammar. The exact start and end lines where a copyright is found are approximate.

The goal of this project is to refactor Copyright detection for speed and simplicity possibly implementing a new parser (PEG?, etc.) or re-implementing core elements in Rust with a Python binding for speed or using a fork of NLTK or any other tool to be faster and more accurate.

This would include also keeping track of line numbers and offsets where copyrights are found.

Also we detect copyrights that are part of a standard license text (e.g. FSF copyright in a GPL text) and we should be able to filter these out.

- **Level**
    - Advanced
- **Tech**
    - Python, Rust, Go?
- **URLS**
    - https://github.com/nexB/scancode-toolkit/tree/develop/src/cluecode
- **Mentors**
    - @JonoYang https://github.com/JonoYang

Port ScanCode to Python 3

ScanCode runs only on Python 2.7 today. The goal of this project is to port ScanCode to support both Python 2 and Python 3.

- **Level**
    - Intermediate to Advanced
- **Tech**
    - Python, C/C++, Go (for native code)
- **URLS**
    - https://github.com/nexB/scancode-toolkit/issues/295
- **Mentors**
    - @steven-esser https://github.com/steven-esser

Improve Programming language detection and classification in ScanCode

ScanCode programming language detection is not as accurate as it could be and this is important to get this right to drive further automation. We also need to automatically classify each file in facets when possible.

The goal of this project is to improve the quality of programming language detection (which is using only Pygments today and could use another tool, e.g. some Bayesian classifier like Github linguist, enry ?). And to create and implement a flexible framework of rules to automate assigning files to facets which could use some machine learning and classifier.

- **Level**
    - Intermediate to Advanced
- **Tech**
    - Python
- **URLS**
    - https://github.com/nexB/scancode-toolkit/issues/426
    - https://github.com/nexB/scancode-toolkit/issues/1012
    - https://github.com/nexB/scancode-toolkit/issues/1036
- **Mentors**
    - @pombredanne https://github.com/pombredanne

Improve License detection accuracy and speed in ScanCode

ScanCode license detection is using a sophisticated set of techniques base on automatons, inverted indexes and sequence matching. There are some cases where license detection accuracy could be improved (such as when scanning long notices). Other improvements would be welcomed to ensure the proper detected license text is collected in an improved way. Dealing with large files sometimes trigger a timeout and handling these cases would be needed too (by breaking files in chunks). The detection speed could also be improved possibly by porting some critical code sections to C or Rust and that would need extensive profiling.

- **Level**
    - Advanced
- **Tech**
    - Python, C/C++, Rust, Go
- **Mentors**
    - @mjherzog https://github.com/mjherzog
    - @pombredanne https://github.com/pombredanne

Improve ScanCode scan summarization and deduction

The goal of this project is to take existing scan results and infer summaries and perform some deduction of license and origin at a higher level, such as the licensing or origin of a whole directory tree. The ultimate goal is to automate the conclusion of a license and origin based on scans. This could include using statistics and machine learning techniques such as classifiers where relevant and efficient.

This should be implemented as a set of ScanCode plugins and further the summarycode module plugins.

- **Level**
  - Advanced
- **Tech**
  - Python (Rust and Go welcomed too)
- **URLS**
  - https://github.com/nexB/scancode-toolkit/issues/426
  - https://github.com/nexB/scancode-toolkit/issues/377
- **Mentors**
  - @pombredanne https://github.com/pombredanne
  - @JonoYang https://github.com/JonoYang

Create Linux distros and FreeBSD packages for ScanCode.

The goal of this project is to ensure that we have proper packages for Linux distros and FreeBSD for ScanCode.

The first step is to debundle pre-built binaries that exist in ScanCode such that they come either from system-packages or pre-built Python wheels. This covers libarchive, libmagic and a few other native libraries and has been recently completed.

The next step is to ensure that all the dependencies from ScanCode are also available as distro packages.

The last step is to create proper distro packages for RPM, Debian, FreeBSD and as many other distros such as Nix and GUIX, Alpine, Arch and Gentoo (and possibly also AppImage.org packages and Docker images) and submit these package to the distros.

As a bonus, the same could then be done for AboutCode toolkit and TraceCode.

This requires a good understanding of packaging and Python.

- **Level**
  - Intermediate to Advanced
- **Tech**
  - Python, Linux, C/C++ for native code
- **URLS**
  - https://github.com/nexB/scancode-toolkit/issues/487
  - https://github.com/nexB/scancode-toolkit/issues/469
- **Mentor**
  - @pombredanne https://github.com/pombredanne

DeltaCode projects

Approximately Similar file detection in DeltaCode

DeltaCode is a tool to compare and report scan differences. When comparing files, it only uses exact comparison. The goal of this project is to improve the usefulness of the delta by also finding files that are mostly the same (e.g. quasi or nrea duplicates) vs. files that are completely different. Then the DeltaCode comparison core should be updated accordingly to detect and report material changes to scans (such as new, update or removed licenses, origins and packages) when changes are also meterial in the code files (e.g. such that small changes may be ignored)

- **Level**
  - Intermediate to Advanced
- **Tech**
  - Python
- **URLS**
  - https://github.com/nexB/deltacode/
- **Mentors**
  - @steven-esser https://github.com/steven-esser
  - @johnmhoran https://github.com/johnmhoran

TraceCode projects

Static analysis of binaries for build tracing in TraceCode

TraceCode does system call tracing only today. The primary goal of this project is to create a tool that provides the same results as the strace-based tracing but would be using using ELF symbols, DWARF debug symbols, signatures or string matching to determine when and how a source code file is built in a binary using only a static analysis. The primary target should be Linux executables, though the code should be designed to be extensible to Windows PE and macOS Dylib and exes.

- **Level**
  - Advanced
- **Tech**
  - Python, Linux, ELFs, DWARFs, symbols, reversing
- **URLS**
  - https://github.com/nexB/tracecode-toolkit for the existing non-static tool
  - https://github.com/nexB/scancode-toolkit-contrib for some work in progress on binaries/symbols parsers/extractors
- **Mentor**
  - @pombredanne https://github.com/pombredanne

Improve dynamic build tracing in TraceCode

TraceCode does system call tracing and relies on kernel-space system calls and in particular tracing file descriptors. This project should improve the tracing of the lifecycle of file descriptors when tracing a build with strace. We need to improve how TraceCode does system call tracing by improving the way we track open/close file descriptors in the trace to reconstruct the lifecycle of a traced file. This requires to understand and dive if the essence of system calls and file lifecycle from a kernel point of view and build datastructure and code to reconstruct user-space file activity from the kernel traces along a timeline.

This project also would cover updating TraceCode to use the Click command line toolkit (like for ScanCode).

- **Level**
  - Advanced
- **Tech**
  - Python, Linux kernel, system calls
- **URLS**
  - https://github.com/nexB/tracecode-toolkit for the existing non-static tool
  - https://github.com/nexB/scancode-toolkit-contrib for the work in progress on binaries/symbols parsers/extractors
- **Mentor**
  - @pombredanne https://github.com/pombredanne

Conan and Other projects

Containers and VM images static package analysis

The goal of this project is to further the Conan container static analysis tool to effectively support proper inventory of installed packages without running the containers.

This includes determining which packages are installed in Docker layers for RPMs, Debian or Alpine Linux in a static way. And this may eventually require the integration with ScanCode.

- **Level**
  - Advanced
- **Tech**
  - Python, Go, containers, distro package managers, RPM, Debian, Alpine
- **URLS**
  - https://github.com/nexB/conan
- **Mentor**
  - @JonoYang https://github.com/JonoYang

DependentCode: a mostly universal Package dependencies resolver

The goal of this project is to create a tool for a universal package dependencies resolution using a SAT solver that should leverage the detected packages from ScanCode and the Package URLs and could provide a good enough way to resolve package dependencies for many system and application package formats. This is a green field project.

- **Level**
    - Advanced
- **Tech**
    - Python, C/C++, Rust, SAT
- **URLS**
    - https://github.com/package-url
    - https://fosdem.org/2018/schedule/event/purl/
    - https://github.com/heremaps/oss-review-toolkit
- **Mentors**
    - @pombredanne https://github.com/pombredanne

VulnerableCode Package security vulnerability correlated data feed

This project is to futher and evolve the VulnerableCode server and software package vulnerabilities data aggregator.

VulnerableCode was started as a GSoC project in 2017. Its goal is to collect, aggregate and correlate vulnerabilities data and provide semi-automatic correlation. In the end it should provide the basis to report vulnerabilities alerts found in packages identified by ScanCode.

This is not trivial as there are several gaps in the CVE data and how they relate to packages as they are detected by ScanCode or else.

The features and TODO for this updated server would be:

- Aggregate more and new packages vulnerabilities feeds,
- Automating correlation: add smart relationship detection to infer new relatiosnhips between available packages and vulnerabilities from mining the graph of existing relations.
- Create a ScanCode plugin to report vulnerabilities with detected packages using this data.
- Integrate API lookup on the server withe the AboutCode Manager UI
- Create a UI and model for community curation of vulnerability to package mappings, correlations and enhancements.
- **Level**
    - Advanced
- **Tech**
    - Python, Django
- **URLS**
    - https://github.com/nexB/vulnerablecode
    - https://github.com/nexB/aboutcode-manager
    - https://github.com/nexB/scancode-toolkit

- Other interesting pointers:

  * https://github.com/cve-search/cve-search

  * https://github.com/jeremylong/DependencyCheck/

  * https://github.com/victims/victims-cve-db

  * https://github.com/rubysec/ruby-advisory-db

  * https://github.com/future-architect/vuls

  * https://github.com/coreos/clair

  * https://github.com/anchore/anchore/

  * https://github.com/pyupio/safety-db

  * https://github.com/RetireJS/retire.js

  * and many more including Linux distro feeds

- **Mentors**

  - @steven-esser https://github.com/steven-esser

  - @JonoYang https://github.com/JonoYang

High volume matching automatons and data structures

Finding similar code is a way to detect the origin of code against an index of open source code.

To enable this, we need to research and create efficient and compact data structures that are specialized for the type of data we lookup. Given the volume to consider (typically multi billion values indexed) there are special considerations to have compact and memory efficient dedicated structures (rather than using a general purpose DB or Key/value pair store) that includes looking at automata, and memory mapping. This types of data structures should be implemented in Rust as a preference (though C/C++ is OK) and include Python bindings.

There are several areas to research and prototype such as:

- A data structure to match efficiently a batch of fix-width checksums (e.g. SHA1) against a large index of such checksums, where each checksum points to one or more files or packages. A possible direction is to use finite state transducers, specialized B-tree indexes, blomm-like filters. Since when a codebase is being matched there can be millions of lookups to do, the batch matching is preferred.

- A data structure to match efficiently a batch of fix-width byte strings (e.g. LSH) against a large index of such LSH within a fixed hamming distance, where each points to one or more files or packages. A possible direction is to use finite state transducers (possibly weighted), specialized B-tree indexes or multiple hash-like on-disk tables.

- A memory-mapped Aho-Corasick automaton to build large batch tree matchers. Available Aho-Corasick automatons may not have a Python binding or may not allow memory-mapping (like pyahocorasick we use in ScanCode). The volume of files we want to handle requires to reuse, extend or create specialized tree/paths matching automatons that can handle eventually billions of nodes and are larger than the available memory. A possible direction is to use finite state transducers (possibly weighted).

- Feature hashing research: we deal with many "features" and hashing to limit the number and size of the each features seems to be a valuable thing. The goal is to research the validaty of feature hashing with short hashes (15, 16 and 32 bits) and evaluate if this leads to acceptable false-positive and loss of accuracy in the context of the data structures mentioned above.

Then using these data structures, the project should create a system for matching code as a Python-based server exposing a simple API. This is a green field project.

- **Level**
    - Advanced
- **Tech**
    - Rust, Python
- **URLS**
    - https://github.com/nexB/scancode-toolkit-contrib for samecode fingerprints drafts.
    - https://github.com/nexB/scancode-toolkit for commoncode hashes
- **Mentors**
    - @pombredanne https://github.com/pombredanne

Mentoring

We welcome new mentors to help with the program and require some good unerstanding of the project codebase and domain to join as a mentor. Contact the team on Gitter.

Google Summer of Code 2018

See *Contributor Project Ideas (old)*.

Google Summer of Code 2017

Welcome to AboutCode! This year AboutCode is a mentoring Organization for the Google Summer of Code 2017 edition.

AboutCode is a project to uncover data … about software code:

- where does it come from?
- what is its license? copyright?
- is it secure, maintained, well coded?

All these are questions that are important to find answers to when there are million of free and open source software components available on the web.

Where software comes from and what is its license should be a problem of the past, such that everyone can safely consume more free and open source software. Come and join us to make it so!

Our tools are used to help detect and report the origin and license of source code, packages and binaries, as well as discover software and package dependencies, track vulnerabilities, bugs and other important software component attributes.

Contact

Subscribe to the mailing list at https://lists.sourceforge.net/lists/listinfo/aboutcode-discuss and introduce yourself and start the discussion! The mailing list is usually the better option to avoid timezone gaps.

The list archive have also plenty of interesting information. Someone may have asked your question before. Search and browse the archives at https://sourceforge.net/p/aboutcode/mailman/aboutcode-discuss/ !

For short chats, you can also join the #aboutcode IRC channel on Freenode or the Gitter channel at https://gitter.im/aboutcode-org/discuss

For personal issues, you can contact the org admin directly: @pombredanne and pombredanne@gmail.com

Please ask questions the smart way: http://www.catb.org/~esr/faqs/smart-questions.html

Technology

Discovering the origin of code is a vast topic. We primarily use Python for this and some C/C++ and JavaScript, but we are open to using any other language within reason.

Our domain includes text analysis and processing (for instance for copyrights and licenses), parsing (for package manifest formats), binary analysis (to detect the origin and license of binaries, which source code they come from, etc) as well as web based tools and APIs (to expose the tools and libraries as web services).

About your project application

We expect your application to be in the range of 1000 words. Anything less than that will probably not contain enough information for us to determine whether you are the right person for the job. Your proposal should contain at least the following information, plus anything you think is relevant:

- Your name
- Title of your proposal
- Abstract of your proposal
- Detailed description of your idea including explanation on why is it innovative and what it will contribute
    - hint: explain your data structures and the main processing flows in details.
- Description of previous work, existing solutions (links to prototypes, bibliography are more than welcome)
- Mention the details of your academic studies, any previous work, internships
- Relevant skills that will help you to achieve the goal (programming languages, frameworks)?
- Any previous open-source projects (or even previous GSoC) you have contributed to and links.
- Do you plan to have any other commitments during GSoC that may affect your work? Any vacations/holidays? Will you be available full time to work on your project? (Hint: do not bother applying if this is not a serious full time commitment)

Subscribe to the mailing list at https://lists.sourceforge.net/lists/listinfo/aboutcode-discuss or join the #aboutcode IRC channel on Freenode and introduce yourself and start the discussion!

You need to understand something about open source licensing or package managers or code and binaries static analysis. The best way to demonstrate your capability would be to submit a small patch ahead of the project selection for an existing issue or a new issue.

Project ideas

ScanCode live scan server :

This project is to use ScanCode as a library in a web and REST API application that allows you to scan code on demand by entering a URL and then store the scan results. It could also be made available as a Travis or Github integration to scan on commit with webhooks. Bonus feature is to scan based on a received tweet of similar IRC or IM integration.

- **URLS** :
    - https://github.com/nexB/scancode-toolkit
- **Mentors** :
    - @steven-esser https://github.com/steven-esser
    - @tdruez https://github.com/tdruez

Package security vulnerability data feed (and scanner) :

The end goal for this project is to build on existing projects to match packages identified by ScanCode to existing vulnerability alerts. This is not trivial as there are several gaps in the CVE data and how they relate to packages as they are detected by ScanCode or else. This is a green field project.

The key points to tackle are:

1. create the tools to build a free and open source structured and curate security feed

    - the aggregation of packages vulnerabilities feeds in a common and structured model (CVE, distro trackers, etc),

    - the aggregation of additional security data (CWE, CPE, and more) in that model,

    - the correlation of the different data items, creating accurate relationships and matching of actual package identifiers to vulnerabilities,

    - an architecture for community curation of vulnerabilities, correlation and enhancement of the data.

2. as a side bonus, build the tools in ScanCode to match detected packages to this feed. Note there is no FOSS tool and DB that does all of this today (only proprietary solutions such as vfeed or vulndb).

- **Some Related URLS for other projects in the same realm** :
    - https://github.com/cve-search/cve-search
    - https://github.com/jeremylong/DependencyCheck/
    - https://github.com/victims/victims-cve-db
    - https://github.com/rubysec/ruby-advisory-db
    - https://github.com/future-architect/vuls
    - https://github.com/coreos/clair
    - https://github.com/anchore/anchore/
    - https://github.com/pyupio/safety-db
    - https://github.com/RetireJS/retire.js
    - and many more including Linux distro feeds
- **Mentors** :

– @steven-esser https://github.com/steven-esser

– @JonoYang https://github.com/JonoYang

– @pombredanne https://github.com/pombredanne

Port the Python license expression library to JScript and prepare and publish an NPM package:

Use automated code translation (for JS) for the port. Add license expression support to AboutCodeMgr with this library. As a bonus, create a web server app and API service to parse and normalize ScanCode and SPDX license expressions either in Python or JavaScript.

- **URLS** :

    – https://github.com/nexB/license-expression

    – https://github.com/bastikr/boolean.py

    – https://github.com/nexB/aboutcode-manager

    – https://github.com/jashkenas/coffeescript/wiki/list-of-languages-that-compile-to-js

- **Mentors** :

    – @JonoYang https://github.com/JonoYang

    – @steven-esser https://github.com/steven-esser

MatchCode :

Create a system for matching code using checksums and fingerprints against a repository of checksums and fingerprints. Create a basic repository for storing these fingerprints and expose a basic API. Create a client that can collect fingerprints on code and get matches using API calls to this repository or package manager APIs (Maven, Pypi, etc), or search engines APIs such as searchcode.com, debsources, or Github or Bitbucket commit hash searches/API or the SoftwareHeritage.org API.

- **URLS** :

    – https://github.com/nexB/scancode-toolkit-contrib for samecode fingerprints drafts.

    – https://github.com/nexB/scancode-toolkit for commoncode hashes

- **Mentors** :

    – @pombredanne https://github.com/pombredanne

ScanCode scan deduction :

The goal of this project is to take existing scan and match results and infer summaries and deduction at a higher level, such as the licensing of a whole directory tree.

- **URLS** :

    – https://github.com/nexB/scancode-toolkit/issues/426

    – https://github.com/nexB/scancode-toolkit/issues/377

- **Mentors** :

    – @pombredanne https://github.com/pombredanne

– @JonoYang https://github.com/JonoYang

DeltaCode :

A new tool to help determine at a high level if the licensing for two codebases or versions of code has changed, and if so how. This is NOT a generic diff tool that identifies all codebase differences, rather it focuses on changes in licensing based on differences between ScanCode files.

- **Mentor** :

  – @steven-esser https://github.com/steven-esser

License and copyright detection benchmark :

Compare ScanCode runtimes with Fossology, licensee, LicenseFinder, license-check, ninka, slic, LiD and others. This project is to create a comprehensive test suite and a benchmark for several FOSS open source license and copyright detection engines, establish mappings between the different conventions they use for license identification and evaluate and publish the results of detection accuracy and precision.

Note that this not about the speed of scanning: the performance and time taken is accessory and a nice to have result only. What matters is the accuracy of the license detection:

1. is the right license detected and how correct is this detection?

2. when a license is detected is the correct exact text matched and returned?

So what is needed is a (large) test set of files.

Then establishing a ground truth for reference e.g. detecting then reviewing manually possibly with scancode to set up the baseline that will be used to compare all the scanners.

Then run the other tools and scancode to see how well they perform and of course establish a mapping of license identifiers: each tool may use different license ids so we need to map these to the ids used in the test baseline (e.g. the scancode license keys): all this has to be built, possibly reusing some or all of the scancode tests and lacing in all the tests from the other tools and adding more ass needed.

- **Mentors** :

  – @mjherzog https://github.com/mjherzog

  – @pombredanne https://github.com/pombredanne

Improved copyright parsing in ScanCode :

by keeping track of line numbers and offsets where copyrights are found. This would likely require either replacing or enhancing NLTK which is used as a natural language parser to add support for tracking where a copyright has been detected in a scanned text.

- **URLS** :

  – https://github.com/nexB/scancode-toolkit/tree/develop/src/cluecode

- **Mentor** :

  – @JonoYang https://github.com/JonoYang

Support full JSON and ABCD formats in AttributeCode

- **URLS** :

    – https://github.com/nexB/attributecode/issues/277

- **Mentor** :

    – @chinyeungli https://github.com/chinyeungli

Transparent archive extraction in ScanCode :

ScanCode archive extraction is currently done with a separate command line invocation. The goal of this project is to integrate archive extraction transparently into the ScanCode scan loop.

- **URLS** :

    – https://github.com/nexB/scancode-toolkit/issues/14

- **Mentor** :

    – @pombredanne https://github.com/pombredanne

Automated docker and VM images static package analysis :

to determine which packages are installed in Docker layers for RPMs, Debian or Alpine Linux. This is for the conan Docker image analysis tool.

- **URLS** :

    – https://github.com/pombredanne/conan

- **Mentor** :

    – @pombredanne https://github.com/pombredanne

Plugin architecture for ScanCode :

Create ScanCode plugins for outputs to multiple formats (CSV, JSON, SPDX, Debian Copyright)

- **URLS** :

    – https://github.com/nexB/scancode-toolkit/issues/552

    – https://github.com/nexB/scancode-toolkit/issues/381

- **Mentor** :

    – @pombredanne https://github.com/pombredanne

Static analysis of binaries for build tracing in TraceCode :

TraceCode does system call tracing. The goal of this project is to do the same using symbol, debug symbol or string matching to accomplish something similar,

- **URLS** :
    - https://github.com/nexB/tracecode-build for the existing non-static tool
    - https://github.com/nexB/scancode-toolkit-contrib for the work in progress on binaries/symbols parsers/extractors
- **Mentor** :
    - @pombredanne https://github.com/pombredanne

Better support tracing the lifecycle of file descriptors in TraceCode build :

TraceCode does system call tracing. The goal of this project is to improve the way we track open/close file descriptors in the trace to reconstruct the life of a file.

- **URLS** :
    - https://github.com/nexB/tracecode-build
- **Mentor** :
    - @pombredanne https://github.com/pombredanne

Create Debian and RPM packages for ScanCode, AttributeCode and TraceCode.

Consider also including an AppImage.org package. If you think this may not fill in a full three months project, consider also adding some extras such as submitting the packages to Debian and Fedora.

- **URLS** :
    - https://github.com/nexB/scancode-toolkit/issues/487
    - https://github.com/nexB/scancode-toolkit/issues/469
- **Mentor** :
    - @pombredanne https://github.com/pombredanne

AboutCode Manager test suite and Ci :

Create an extensive test suite for the Electron app and setup the CI to run unit, integration and smoke tests on Ci for Windows, Linux and Mac.

- **URLS** :
    - https://github.com/nexB/aboutcode-manager
- **Mentors** :
    - @jdaguil https://github.com/jdaguil
    - @pombredanne https://github.com/pombredanne

DependentCode :

Create a tool for mostly universal package dependencies resolution.

- **URLS** :
    - https://github.com/nexB/dependentcode
- **Mentors** :
    - @pombredanne https://github.com/pombredanne

FetchCode :

Create a tool for mostly universal package and code download from VCS, web, ftp, etc.

- **Mentors** :
    - @pombredanne https://github.com/pombredanne

2.3 GSoD – Google Season of Docs

Google Season of Docs is an international annual program giving technical writers an opportunity to contribute to open source projects.It fosters the open source contributions of technical writers. GSoD was started by Google in 2019. More about of GSoD - <https://developers.google.com/season-of-docs>_

2.3.1 GSoD 2021

scancode.io Documentation GSoD Case Study

AboutCode.org is a community of open source developers who are trying to make open source easier to use by providing open source tools to discover, identify and track open source components (aka. Software Composition Analysis - SCA). AboutCode is the collective name for these open source tools.

Author: *Dennis Clark https://github.com/DennisClark*

Problem Statement

Create and update ScanCode.io HowTo Guides and Tutorials

Proposal Abstract

Project Proposal Link

Our focus for GSoD 2021 is Scancode.io and these specific tasks:

- Create a HowTo guide for integrating third-party libraries into a ScanCode.io Pipeline.
- Add a tutorial for adding a new Pipeline with a third-party library.
- Extend the HowTo Guides to cover Software Composition Analysis workflows
- Upgrade the ScanCode.io Web UI documentation

- Create an introductory video to show how the web UI is used.

- Update and improve the existing Pipe libraries reference API documentation (which is generated from code documentation "docstrings").

- Sync the new documentation set with the code to support continuous integration.

Project Description

Creating the proposal

ScanCode.io is an open platform to script and automate Software Composition Analysis (SCA) for many different use cases including development codebases, package dependencies, and Docker, Virtual Machine images, or other containers. With the ScanPipe feature of ScanCode.io, you can create any number of Pipelines to integrate SCA into your development process. A Pipeline may include libraries beyond those available from AboutCode.org. This flexibility means that we need much more extensive HowTo, Tutorials, and Reference documentation to help our users build their own Pipelines. ScanCode.io also offers a Web UI that needs a new Tutorial and HowTo documentation.

Although we already offer a few Pipeline templates, our users need to be self-sufficient to adapt these templates or add new Pipelines in order to fit SCA into their own environment. We also want to encourage users to contribute back improvements to our templates or new Pipeline templates. Both of these objectives require more and better documentation.

Budget

Our budget for the project was $15,000 which was allocated completely to the technical writer working on the project.

Participants

The primary participant was technical writer Hanan Younes who responded to our announcement describing the project scope, and was the clear choice among the many responders based on multiple criteria including clarity of vision and the ability to execute the project goals.

The primary mentor is a regular contributor to AboutCode.org projects and the primary maintainer of ScanCode.io:

- Thomas Druez

Additional mentoring was provided by:

- Philippe Ombredanne

- Michael Herzog

- Dennis Clark

- Ayan Sinha Mahapatra

As an exceptionally thoughtful and articulate analyst, Hanan Younes offered much useful feedback and many questions that encouraged the project mentors to improve their planning and organization. We learned how valuable a clear roadmap and directions can be to a technical writer.

Timeline

We planned that the technical writer would work on our project to ½ time over a period of 5 months (May to October, 2021). Results

The documentation for ScanCode.io was dramatically improved. See scancode.io RTD

The documentation is now well organized into Getting Started, Tutorial, and Reference sections, each of which provides a wealth of information.

Tracking

We used GitHub Projects to manage and track the project. All expectations were met or exceeded.

Analysis

- Hanan executed all tasks beautifully.

- We found that we needed to dig deeper into ScanCode.io to respond to Hanan's many questions and comments.

- The project was very successful and provided everyone with the opportunity to grow and learn. We are very pleased with the documentation that is now available to the open source community.

Summary

- Clear organization of project documentation is essential.

- We may wish to articulate more "stretch goals" in future projects.

- Be sure to define exactly what you want to accomplish.

- Be sure to plan project communication standards, such as a Gitter channel and regularly scheduled status meetings.

Appendix

We received more than twenty proposals, and are grateful to the interest shown in AboutCode.org by the open source community.

Process Feedback to Google

The overall process for the GSoD went well, but we faced a few issues, mostly self-inflicted that we are listing here for reference:

- Since we were not sure if our org would be accepted, we started late working on actual recruitment. Recruitment ended up taking much more time than planned. This means we had to scramble when our org was accepted to get our selected candidate on-boarded and starting the project.

- In contrast with the GSoC project, there is no infrastructure for candidates to submit a project proposal. It was not an issue for us, but we felt after the fact that a GSoC-like infrastructure to funnel submissions would have been a nice addition.

- There was no clear contract or agreement in place with our selected candidate and this led to some misunder-standing between us with regard to payments. This ended up being a source of anxiety for our doc writer and a situation we decided to address directly. AboutCode is sponsored by nexB which is a commercial company and nexB accepted to make a donation to the AboutCode OpenCollective as a bridge to provide advanced payments to our doc writer. For future participation, we will manage this process proactively to avoid such ambiguity.

- While we have been proactively managing the process with our doc writer with weekly meetings and continuous discussions in chat channels, we mentors collectively have not been great at providing timely feedback to GSoD administrators about our progress which is something that we would need to improve for the future.

2.3.2 GSoC 2019

scancode-toolkit Documentation GSoD Report

Project Name

The Project Name was **"Reference for Command Line Options in scancode-toolkit and Reorganize the structure of AboutCode documentation at aboutcode.readthedocs.io"**

About Me

Name - Ayan Sinha Mahapatra
Email : ayansmahapatra@gmail.com
Country : India
Links :

- Github
- Twitter
- Website
- Resume

Important Links for this GSoD Project

- GSoD Report
- GSoD Proposal
- ScanCode Toolkit Docs
- Aboutcode Docs

Project Overview

1. **Restructured Documentation hosted at ReadTheDocs:** Outdated Documentation from GitHub Wikis was updated, improved and converted to .RST, to be hosted at aboutcode.readthedocs.io.

2. **Documentation Tests and Roadmaps:** Documentation tests were added to check Documentation Builds, Internal/External Links, and Style Standards. These tests were added in both aboutcode and scancode-toolkit and already existing/new documentation was made compliant of these tests.

3. **Command Line Options Referance:** The entire Command Line Options were listed, explained, and their application elaborated from scratch. This is the main goal and the most important addition to the scancode-toolkit documentation.

4. **Migration of Scancode-Toolkit docs into its own Repository:** After adding the scancode toolkit docs to aboutcode and reviews, this section was migrated to its own repository, to be distributed with scancode-toolkit and with local builds enabled.

5. **[WIP] Improve/Add to Existing Tutorials/How-Tos:** Added two major Tutorials and other Tutorials/How-Tos were also updated to be consistent with the latest scancode versions and more Tutorials/How-Tos can be added later to better help the users.

6. **[WIP] Add Plugin Support:** A cookiecutter based plugin creation system has to be added, and relevant docs for this process and for the plugin architecture has to be added.

7. **[WIP] Solve Other Documentation Issues:** Solving multiple issues listed under the tag "Documentation".

Pull Requests

PRs before GSoD Started @ nexB/aboutcode

- [#17] Adds Travis-CI for sphinx-build and linkcheck

- [#15] Adds Documentation from Wiki

- [#13] Adds GitHub wiki of Scancode-Toolkit and Scancode-Workbench

GSoD PRs @ nexB/scancode-toolkit

- [#1784] Add Travis CI Test For Docs

- [#1827] Add New Issue and Pull Request Templates

- [#1812] Add Documentation from Aboutcode

- [WIP] Add Cookiecutter Plugin and Plugin Docs

- [WIP] [#1847] Fix remaining documentation Issues

GSoD PRs @ nexB/aboutcode

- [#26] Adds Documentation Versioning, Command Line Options, Changelog
- [#27] CLI Options Reference for v3.1.1
- [#29] Update Docs with changes in Wiki
- [#30] Add doc8 style checks to Travis-CI
- [#31] Fix template error
- [#33] Add final Changes/Improvements before Migrating
- [#34] Delete scancode-toolkit documentation
- [#35] Add GSoD 2019 Report

Acknowledgments

I would like to thank Dennis Clark, Philippe Ombredanne and Steven Esser for their help throughout my GSoD project, from helping me prepare my proposal and project to constructive criticism, nit-picky reviews, conference calls to discuss project roadmaps and to explain to me when I was stuck in any problem. Their support and positivity helped me through this project and made me enjoy every bit of it.

2.3.3 GSoD project ideas pages

Google Season of Docs 2021

AboutCode.org is applying to be a mentoring organization for the Google Season of Docs (GSoD) 2021.

In the event that our proposal is accepted, we'll be looking for fellow members of the FOSS community with technical writing skills and an interest in helping. If you're interested, keep reading to get a sense of what we have in mind for GSoD 2021.

Contents

- *AboutCode Overview*
  - *AboutCode Projects*
  - *Technology*
- *Technical Skills Needed*
- *About Your Project Application*
  - *Create and update scancode.io user and developer guides*
  - *Create ScanCode-Toolkit Plugin Author Guide*
- *Mentoring*
  - *Contact Info*

AboutCode Overview

AboutCode.org is a community of developers behind a suite of Software Composition Analysis tools (command line tools, web-based and API servers and desktop applications) and data for license, origin and security:

- Where does the code come from?
- What software packages are present?
- What is its license? copyright?
- Is the code secure, maintained, well coded?
- Are there any known vulnerabilities?

All these questions are important, and are relevant to millions of free and open source software components available on the web for reuse. The answers are critical to ensure that everyone can safely consume free and open source software.

Join us to make it so!

AboutCode Projects

Our focus for GSoD 2021 is to improve documentation on our flagship projects:

- ScanCode Toolkit
- ScanCode.io
- Vulnerablecode

And specifically to create/update developement and user guides.

See *AboutCode Project Overview* for a complete list of AboutCode projects.

Technology

We use Python for programming AboutCode software. Our projects are a combination of command line tools, libraries, and web based applications based on Django and JavaScript.

Our command line tools are designed to run on Linux, macOS and Windows, and our web-based applications run only on Linux.

We write our documentation using reStructuredText with Sphinx and ReadTheDocs.

We use git and GitHub to store our code and track tasks and issues.

Technical Skills Needed

Incoming technical writers will need the following skills:

- Basic understanding of reStructuredText, Sphinx and ReadTheDocs-based documentation toolchain including git.
- Ability to understand and run programs from the command line in a terminal.
- Ability to read well-structured Python code, and to update Python docstrings.
- Ability to manage Windows and Linux based installation of our tools if need be in virtual machines.
- Ability to write technical documentation such as Tutorials, How-To Guides, and API References.

We do not expect a working knowledge of our tools as this will be picked from mentors and community guidance. An interest in FOSS licensing and software code and origin analysis would be welcome but is not a hard requirement.

Some of these skills could be picked up during the project with help from your mentors, for candidates with strong writing skills.

About Your Project Application

Check out the GSoD Tech Writer Guide
    and Statement Template.

Your statement of interest should be in the range of 1,000 words, and should contain the following information, plus anything else that you think is relevant:

- Personal information, i.e. Your name and contact details.

- A Project statement.

- A detailed description of your project.

- Description of your relevant skills.

- Professional information: Description of previous work, existing solutions, open-source projects, preferably with links.

- Proposed Timeline

- Proposed total budget

- Do you plan to have any other commitments during GSoD that may affect your work? Any vacations/holidays? Will you be available full time to work on your project? Please apply only if this is a serious full time commitment during the GSoD time frame.

Join the chat online or by IRC at https://gitter.im/aboutcode-org/gsod-season-of-docs. Introduce yourself and start a discussion!

An excellent, competitive way to demonstrate your capability would be to submit a documentation improvement (small changes/typos are welcome but do not demonstrate your ability to write documentation) to an AboutCode project, especially to ScanCode Toolkit, Scancode.io.

Please select a project from the list below.

Create and update scancode.io user and developer guides

- Create an HowTo guide for integrating third-party libraries into a ScanCode.io Pipeline.

- Add a tutorial for adding a new Pipeline with a third-party library.

- Extend the HowTo Guides to cover Software Composition Analysis workflows based on ScanCode.io.

- Upgrade the scancode.io Web UI documentation.

- Create an introductory video to teach how the web UI is used.

- Update and improve the existing Pipe libraries reference API documentation (which is generated from code documentation "docstrings").

- Sync the new documentation set with the code to support continuous integration with code changes.

- **Level**

- – Intermediate

- **Tech**

  - – Command line processing in a Linux-compatible terminal window

  - – Python [Django]

  - – GitHub, reStructured Text and Sphinx

- **URLs**

  - – https://scancodeio.readthedocs.io

  - – https://github.com/nexB/scancode.io

  - – https://gitter.im/aboutcode-org/discuss

- **Mentors**

  - – https://github.com/johnmhoran

  - – https://github.com/mjherzog

  - – https://github.com/tdruez

  - – https://github.com/AyanSinhaMahapatra

Create ScanCode-Toolkit Plugin Author Guide

The goal of this guide is to provide an end-to-end tutorial and instruction set on how to create scancode-toolkit plugins, with concrete examples and to generate a reference API documentation for modules, classes and functions used to create plugins.

Some of the specific supported activities would be:

- Explain the Plugin Architechture used in scancode-toolkit

- Explain the types of plugins used in scancode-toolkit (pre-installed/seperately-installed) and (post-scan, pre-scan, scanner)

- An exhaustive list of all plugins and what they do, with links to their code.

- How to create a minimal plugin adding some functionality to scancode-toolkit.

- Useful examples of different types of plugins

- Using nexb/skeleton as a template for plugins.

Note: Filter current scancode-toolkit GitHub Issues by Label 'documentation' to see many examples of specific areas for improvement.

- **Level**

  - – Intermediate

- **Tech**

  - – Some lightweight python programming to create example plugins and understand how they are constructed, the mentors will help here.

  - – Command line processing in a Linux-compatible terminal window

  - – GitHub, reStructured Text and Sphinx

- **URLs**

- https://scancode-toolkit.readthedocs.io

- https://github.com/nexB/scancode-toolkit

- https://gitter.im/aboutcode-org/scancode

- **Mentors**

  - https://github.com/johnmhoran

  - https://github.com/steven-esser

  - https://github.com/AyanSinhaMahapatra

Mentoring

We have an established team of contributors that are willing to mentor the selected Tech Writers, and they have extensive experience from mentoring in the open, including a successful previous participation in Google Season of Docs 2019. We have a developed a gentle and efficient way to mentor our contributors to success.

Contact Info

Join the chat online or by IRC at https://gitter.im/aboutcode-org/gsod-season-of-docs. Introduce yourself and start the discussion!

You can also contact:

- the primary org admin directly: @pombredanne and pombredanne@gmail.com

- the GSoD coordinator directly: jmhoran@nexb.com

Google Season of Docs 2020

AboutCode.org is applying to be a mentoring organization for the Google Season of Docs (GSoD) 2020.

In the event that our application is accepted, we'll be looking for fellow members of the FOSS community with technical writing skills and an interest in helping. If you're interested, keep reading to get a sense of what we have in mind for GSoD 2020.

Contents

- – *Improve ScanCode Workbench (SCWB) Documentation*

  – *Automate Code Documentation*

- *Your Documentation Project Ideas*

- *Mentoring*

AboutCode Overview

AboutCode is a family of FOSS projects for Software Composition Analysis (SCA) and FOSS Compliance:

- Where does code come from?

- What is its license?

- What is its origin and copyright?

- Is the code secure, maintained, well coded?

All of these questions are important, and are relevant to millions of free and open source software components available on the web for reuse. The answers are critical to ensure that anyone and everyone can safely consume free and open source software.

Join us to make it so!

Our tools are used to detect and report the origin and license of source code, packages and binaries as well as to discover software and package dependencies. We are also working on projects to track security vulnerabilities, bugs and other important software package attributes.

AboutCode Projects

Our focus for GSoD 2020 is on ScanCode Toolkit and Scancode Workbench, but proposals to improve the documentation for other AboutCode projects are welcome.

See *AboutCode Project Overview* for a complete list of AboutCode projects.

Technology

We primarily use Python for AboutCode software. There is also some code in C/C++ or other languages. We use Electron/JavaScript for the ScanCode Workbench desktop application.

Our domain includes text analysis and processing (e.g., for copyright and license detection), parsing (for package manifest formats), binary analysis (to detect the origin and license of binaries), web-based tools and APIs (to expose the tools and libraries as web services) and low-level data structures for efficient matching (such as Aho-Corasick and other automata).

For documentation we primarily use reStructured Text files with Sphinx and ReadTheDocs. For command line tools we use text files for help documentation. There is still some project documentation in project wikis, but our goal is to migrate that to ReadTheDocs. For each project there will also be a README and a few other files in the corresponding GitHub repository.

Contact Info

Join the chat online or by IRC at https://gitter.im/aboutcode-org/gsod-season-of-docs. Introduce yourself and start the discussion!

You can also contact:

- the primary org admin directly: @pombredanne and pombredanne@gmail.com

- the GSoD coordinator directly: jmhoran@nexb.com

Technical Writing Skills Needed

Incoming technical writers will need the following skills:

- Ability to install and configure open source code from GitHub.

- Ability to understand and run programs from the command line in a terminal window.

- An interest in FOSS licensing and software code and origin analysis.

During the application process and during a GSoD project with us you will learn how to:

- Design and create documentation as Tutorials, HowTo Guides, Reference or Discussions.

  - See https://documentation.divio.com/introduction/ for more information about this approach.

- Create and edit documentation files in reStructured Text.

- Manage documentation files with Sphinx.

- Publish documentation to ReadTheDocs.

We are happy to help you get up to speed. The more you are able to demonstrate ability and skills in advance, the more likely we are to choose your application!

About Your Project Application

Your application should be in the range of 1,000 words, and should contain the following information, plus anything else that you think is relevant:

- Your name and contact details.

- Title of your proposal.

- Abstract of your proposal.

- Description of your idea including an explanation of what it will contribute to the project.

- Description of previous work, existing solutions, open-source projects, preferably with links.

- Details of your academic studies and any previous internships.

- Description of your relevant skills.

- Do you plan to have any other commitments during GSoD that may affect your work? Any vacations/holidays? Will you be available full time to work on your project? Please apply only if this is a serious full time commitment during the GSoD time frame.

Join the chat online or by IRC at https://gitter.im/aboutcode-org/gsod-season-of-docs. Introduce yourself and start a discussion!

An excellent, competitive way to demonstrate your capability would be to submit a documentation improvement to an AboutCode project, especially to ScanCode Toolkit or ScanCode Workbench.

You can pick a project idea from the list below. You can also submit *Your Documentation Project Ideas*.

Our Documentation Project Ideas

Note that the AboutCode focus for GSoD 2020 is on ScanCode Toolkit and ScanCode Workbench, although proposals to improve the documentation for other AboutCode projects are welcome.

Your Documentation Project Ideas are welcome too! Please chat about them to increase your chances of success!

Improve ScanCode Toolkit (SCTK) Documentation

- Upgrade the ScanCode Toolkit documentation on ReadTheDocs by streamlining and re-organizing it.

- Apply the Tutorial, HowTo, Reference and Discussions framework (or propose alternative).

- Implement process to sync documentation changes with releases.

Note: Filter current SCTK GitHub Issues by Label 'documentation' to see many examples of specific areas for improvement.

- **Level**

  - Intermediate

- **Tech**

  - Command line processing in a Linux-compatible terminal window

  - GitHub, reStructured Text and Sphinx

- **URLs**

  - https://scancode-toolkit.readthedocs.io/en/latest/index.html

  - https://github.com/nexB/scancode-toolkit

  - https://gitter.im/aboutcode-org/scancode

- **Mentors**

  - https://github.com/johnmhoran

  - https://github.com/mjherzog

Improve ScanCode Workbench (SCWB) Documentation

- Upgrade the ScanCode Workbench documentation on ReadTheDocs with a focus on Tutorials and HowTo Guides because this is a GUI application.

- Apply the Tutorial, HowTo, Reference and Discussions framework (or propose alternative).

- Implement process to sync documentation changes with releases.

Note: Filter current SCWB GitHub Issues by Label 'documentation' to see many examples of specific areas for improvement.

- **Level**

- – Intermediate
- **Tech**
  - – GUI design
  - – GitHub, reStructured Text and Sphinx
- **URLs**
  - – https://scancode-workbench.readthedocs.io/en/develop/index.html
  - – https://github.com/nexB/scancode-workbench
  - – https://gitter.im/aboutcode-org/scancode-workbench
- **Mentors**
  - – https://github.com/johnmhoran
  - – https://github.com/steven-esser

Automate Code Documentation

- Identify tools to generate code documentation from Python files.
- Define coding standards necessary to support automated generation of code documentation across AboutCode projects.
- Implement process and tools to automate publication of code documentation for ScanCode Toolkit to ReadThe-Docs using Sphinx and related extensions.
- **Level**
  - – Intermediate
- **Tech**
  - – Python programming
  - – GitHub, reStructured Text and Sphinx
- **URLs**
  - – https://scancode-workbench.readthedocs.io/en/develop/index.html
  - – https://github.com/nexB/scancode-workbench
  - – https://gitter.im/aboutcode-org/scancode-workbench
- **Mentors**
  - – https://github.com/johnmhoran
  - – https://github.com/steven-esser

Your Documentation Project Ideas

Download and install ScanCode Toolkit and Scancode Workbench and try them out. For example, you may try scanning an open source software package in a technology with which you are familiar. What are the documentation weak points?

- Is it difficult to get started? A **Tutorial** documentation opportunity.

- Is it difficult to accomplish a specific objective? A **How-To** documentation opportunity.

- Are the capabilities of the tool too mysterious? Do you want to know more about what you can do with it? A **Reference** documentation opportunity.

- Do you feel that you need to understand its concepts better in order to use it and trust it? Do you want to know more about how the code scanning actually works? A **Discussion** documentation opportunity.

Feel free to propose and describe your own documentation ideas.

Mentoring

We welcome new mentors to help with the program. We require some understanding of the project domain to join as a mentor. Contact the team on Gitter at https://gitter.im/aboutcode-org/gsod-season-of-docs

Google Season of Docs 2019

AboutCode has been accepted as a participant in the Google Season of Documents in 2019 as a mentoring org, and is looking for people with technical writing skills. This page contains information for technical writers and anyone else interested in helping.

AboutCode is a family of FOSS projects to uncover data about software code:

- Where does the code come from? which software package?

- What is its license? copyright?

- Is the code secure, maintained, well coded?

All these questions are important, and are relevant to millions of free and open source software components available on the web for reuse. The answers are critical to ensure that everyone can safely consume free and open source software.

Join us to make it so!

Our tools are used to help detect and report the origin and license of source code, packages and binaries as well as to discover software and package dependencies, and in the future track security vulnerabilities, bugs and other important software package attributes. This is a suite of command line tools, web-based and API servers and desktop applications.

Table of Contents

- *Scan a Codebase and Analyze the Results*
- *How-To ideas*
  - *How To Get the License Clarity Score of a Package*
  - *How To Discover Licensing Issues in a Software Project*
- *Reference ideas*
  - *ScanCode Output Formats*
- *Discussion ideas*
  - *Integrating ScanCode into a Software Development Lifecycle*
- *Your Documentation Project ideas*
- *Mentoring*

List of AboutCode projects

Note that the AboutCode focus for GSOD 2019 is on **ScanCode Toolkit** and **ScanCode Workbench**, although proposals to improve the documents of other AboutCode projects are welcome.

- ScanCode Toolkit is a popular command line tool to scan code for licenses, copyrights and packages, used by many organizations and FOSS projects, small and large.
- Scancode Workbench (formerly AboutCode Manager) is a JavaScript, Electron-based desktop application to review scan results and document your origin and license conclusions.
- Other AboutCode projects are described at https://www.aboutcode.org and https://github.com/nexB/aboutcode

  We also work closely with, contribute to and have co-started several other orgs and projects:

- Package URL is an emerging standard to reference software packages of all types with simple, readable and concise URLs.
- SPDX is the Software Package Data Exchange, a specification to document the origin and licensing of software packages.
- ClearlyDefined is a project to review FOSS software and help FOSS projects to improve their licensing and documentation clarity.

Contact

Join the chat online or by IRC at https://gitter.im/aboutcode-org/discuss Introduce yourself and start the discussion!

For personal issues, you can contact the primary org admin directly: @pombredanne and pombredanne@gmail.com

or the GSOD coordinator directly at dmclark@nexb.com

Please ask questions the smart way: http://www.catb.org/~esr/faqs/smart-questions.html

Technology

We primarily use Python (and some C/C++) for code analysis. We use Electron/JavaScript for GUI.

Our domain includes text analysis and processing (for instance for copyright and license detection), parsing (for package manifest formats), binary analysis (to detect the origin and license of binaries, which source code they come from, etc.) as well as web based tools and APIs (to expose the tools and libraries as web services) and low-level data structures for efficient matching (such as Aho-Corasick and other automata).

Our documentation is provided in text files that support the help function of our command line tools. We also have begun to provide documentation in the Wiki section of some AboutCode projects.

Technical Writing Skills Needed

Incoming technical writers will need the following skills:

- Ability to install and configure open source code from GitHub.

- Ability to understand and run programs from the command line in a terminal window.

- Familiarity with the four document functions described at https://www.divio.com/blog/documentation/

- Ability to create and edit wiki pages with multiple markdown languages.

- An interest in FOSS licensing and software code and origin analysis.

We are happy to help you get up to speed, and the more you are able to demonstrate ability and skills in advance, the more likely we are to choose your application!

About your project application

Your application should be in the range of 1000 words, and should contain the following information, plus anything else that you think is relevant:

- Your name and contact details

- Title of your proposal

- Abstract of your proposal

- Description of your idea including an explanation of what it will contribute to the project, such as the software development life cycle requirements that you expect to help with the documentation improvements.

- Description of previous work, existing solutions, open-source projects, preferably with links.

- Details of your academic studies and any previous internships.

- Descriptions of your relevant skills.

- Do you plan to have any other commitments during GSOD that may affect your work? Any vacations/holidays? Will you be available full time to work on your project? Please apply only if this is a serious full time commitment during the GSOD time frame.

Join the chat online or by IRC at https://gitter.im/aboutcode-org/discuss Introduce yourself and start the discussion!

An excellent, competitive way to demonstrate your capability would be to submit a documentation improvement to an AboutCode project, especially to ScanCode Toolkit or ScanCode Workbench.

You can pick any project idea from the list below. You can also submit *Your Documentation Project ideas*.

Our Documentation Project ideas

Here is a list of candidate project ideas for your consideration, organized by documentation function: **Tutorial** , **How-To** , **Reference** , **Discussion**.

Note that the AboutCode focus for GSOD 2019 is on ScanCode Toolkit and ScanCode Workbench, although proposals to improve the documents of other AboutCode projects are welcome.

Your Documentation Project ideas are welcome too! Please chat about them to increase your chances of success!

Tutorial ideas

Scan a Codebase and Analyze the Results

Provide specific instructions to guide a new user to:

- Scan a somewhat complex sample codebase using scancode-toolkit.

- Import the results into ScanCode Workbench.

- Analyze the scan results.

    - **Level**

        * Intermediate

    - **Tech**

        * Command line processing in a Linux-compatible terminal window

    - **URLS**

        * https://github.com/nexB/scancode-toolkit/blob/develop/README.rst

        * https://github.com/nexB/scancode-toolkit/wiki

        * https://github.com/nexB/scancode-workbench/blob/develop/README.md

        * https://github.com/nexB/scancode-workbench/wiki

    - **Mentors**

        * https://github.com/DennisClark

How-To ideas

How To Get the License Clarity Score of a Package

Explain the recommended scancode-toolkit options to get a license clarity score.

- **Level**

    - Intermediate

- **Tech**

    - Command line processing in a Linux-compatible terminal window

- **URLS**

    - https://github.com/nexB/scancode-toolkit/blob/develop/README.rst

- https://github.com/nexB/scancode-toolkit/wiki

- https://github.com/nexB/scancode-workbench/blob/develop/README.md

- https://github.com/nexB/scancode-workbench/wiki

- **Mentors**

  - https://github.com/DennisClark

How To Discover Licensing Issues in a Software Project

- Explain the recommended scancode-toolkit options to discover licenses.

- Explain how to take advantage of license policy support.

  - **Level**

    * Intermediate

  - **Tech**

    * Command line processing in a Linux-compatible terminal window

  - **URLS**

    * https://github.com/nexB/scancode-toolkit/blob/develop/README.rst

    * https://github.com/nexB/scancode-toolkit/wiki

    * https://github.com/nexB/scancode-workbench/blob/develop/README.md

    * https://github.com/nexB/scancode-workbench/wiki

  - **Mentors**

    * https://github.com/DennisClark

Reference ideas

ScanCode Output Formats

Explain the various ScanCode output formats and their business purposes.

- **Level**

  - Intermediate

- **Tech**

  - Command line processing in a Linux-compatible terminal window

- **URLS**

  - https://github.com/nexB/scancode-toolkit/blob/develop/README.rst

  - https://github.com/nexB/scancode-toolkit/wiki

  - https://github.com/nexB/scancode-workbench/blob/develop/README.md

  - https://github.com/nexB/scancode-workbench/wiki

- **Mentors**

  - https://github.com/DennisClark

Discussion ideas

Integrating ScanCode into a Software Development Lifecycle

Discuss options and techniques to integrate ScanCode into a software development lifecycle workflow:

- During software creation and maintenance.

- During software check-out/check-in.

- During sofware build and test.

    - **Level**

        * Intermediate

    - **Tech**

        * Command line processing in a Linux-compatible terminal window

    - **URLS**

        * https://github.com/nexB/scancode-toolkit/blob/develop/README.rst

        * https://github.com/nexB/scancode-toolkit/wiki

        * https://github.com/nexB/scancode-workbench/blob/develop/README.md

        * https://github.com/nexB/scancode-workbench/wiki

    - **Mentors**

        * https://github.com/DennisClark

Your Documentation Project ideas

Download and install ScanCode Toolkit and ScanCode Workbench and try them out. For example, you may try scanning an open source software package in a technology with which you are familiar. What are the documentation weak points?

- Is it difficult to get started? A **Tutorial** document opportunity.

- Is it difficult to accomplish a specific objective? A **How-To** document opportunity.

- Are the capabilities of the tool too mysterious? Do you want to know more about what you can do with it? A **Reference** document opportunity.

- Do you feel that you need to understand its concepts better in order to use it and trust it? Do you want to know more about how the code scanning actually works? A **Discussion** document opportunity.

Feel free to propose and describe your own documentation ideas.

Mentoring

We welcome new mentors to help with the program. We require some understanding of the project domain to join as a mentor. Contact the team on Gitter at https://gitter.im/aboutcode-org/discuss

2.4 Documentation Guidelines

2.4.1 Document Software Setup

AboutCode documentation is built using Sphinx. See http://www.sphinx-doc.org/en/master/index.html

AboutCode documentation is distributed using "Read the Docs". See https://readthedocs.org/

Individual document files are in reStructuredText format. See http://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html

You create, build, and preview AboutCode documentation on your local machine.

You commit your updates to the AboutCode repository on GitHub, which triggers an automatic rebuild of https://aboutcode.readthedocs.io/en/latest/index.html

2.4.2 Clone AboutCode

To get started, create or identify a working directory on your local machine.

Open that directory and execute the following command in a terminal session:

```
git clone https://github.com/nexB/aboutcode.git
```

That will create an /aboutcode directory in your working directory. Now you can install the dependencies in a virtualenv:

```
cd aboutcode
virtualenv -p /usr/bin/python3.6 docs-venv
source docs-venv/bin/activate
```

Now, the following prerequisites are installed

- Sphinx

- sphinx_rtd_theme (the format theme used by ReadTheDocs)

- docs8 (style linter)

```
pip install Sphinx sphinx_rtd_theme doc8
```

Now you can build the HTML documents locally:

```
cd docs
make html
```

Assuming that your Sphinx installation was successful, Sphinx should build a local instance of the documentation .html files:

```
open build/html/index.html
```

In case this command did not work, for example on Ubuntu 18.04 you may get a message like "Couldn't get a file descriptor referring to the console", try:

```
see build/html/index.html
```

You now have a local build of the AboutCode documents.

2.4.3 Improve AboutCode Documents

Before you begin creating and modifying AboutCode documents, be sure that you understand the basics of reStructuredText as explained at http://www.sphinx-doc.org/en/master/usage/restructuredtext/basics.html

Ensure that you have the latest AboutCode files:

```
git pull
git status
```

Use your favorite text editor to create and modify .rst files to make your documentation improvements.

Review your work:

```
cd docs
make html
open build/html/index.html
```

AboutCode uses Travis-CI to test build status and check links, so run this script at your local system before creating a Pull Request.

```
cd docs
./scripts/sphinx_build_link_check.sh
./scripts/doc8_style_check.sh
```

2.4.4 Share AboutCode Document Improvements

Follow standard git procedures to upload your new and modified files. The following commands are examples:

```
git status
git add source/index.rst
git add source/how-to-scan.rst
git status
git commit -m "New how-to document that explains how to scan"
git status
git push
git status
```

The AboutCode webhook with ReadTheDocs should rebuild the documentation after your Pull Request is Merged.

Refer the Pro Git Book available online for Git tutorials covering more complex topics on Branching, Merging, Rebasing etc.

2.4.5 Continuous Integration

The documentations are checked on every new commit through Travis-CI, so that common errors are avoided and documentation standards are enforced. Travis-CI presently checks for these 3 aspects of the documentation :

1. Successful Builds (By using `sphinx-build`)

2. No Broken Links (By Using `link-check`)

3. Linting Errors (By Using `Doc8`)

So run these scripts at your local system before creating a Pull Request:

```
cd docs
./scripts/sphinx_build_link_check.sh
./scripts/doc8_style_check.sh
```

2.4.6 Style Checks Using `Doc8`

How To Run Style Tests

In the project root, run the following command:

```
$ doc8 --max-line-length 100 docs/source/ --ignore D000
```

A sample output is:

```
Scanning...
Validating...
docs/source/scancode-toolkit/misc/licence_policy_plugin.rst:37: D002 Trailing whitespace
docs/source/scancode-toolkit/misc/faq.rst:45: D003 Tabulation used for indentation
docs/source/scancode-toolkit/misc/faq.rst:9: D001 Line too long
docs/source/scancode-toolkit/misc/support.rst:6: D005 No newline at end of file
========
Total files scanned = 34
Total files ignored = 0
Total accumulated errors = 326
Detailed error counts:
    - CheckCarriageReturn = 0
    - CheckIndentationNoTab = 75
    - CheckMaxLineLength = 190
    - CheckNewlineEndOfFile = 13
    - CheckTrailingWhitespace = 47
    - CheckValidity = 1
```

Now fix the errors and run again till there isn't any style error in the documentation.

What is Checked?

PyCQA is an Organization for code quality tools (and plugins) for the Python programming language. Doc8 is a sub-project of the same Organization. Refer this README for more details.

What is checked:

- invalid rst format - D000

- lines should not be longer than 100 characters - D001

    – RST exception: line with no whitespace except in the beginning

    – RST exception: lines with http or https URLs

    – RST exception: literal blocks

    – RST exception: rst target directives

- no trailing whitespace - D002

- no tabulation for indentation - D003

- no carriage returns (use UNIX newlines) - D004

- no newline at end of file - D005

2.4.7 Intersphinx

Aboutcode documentation uses Intersphinx to create links to other Sphinx Documentations, to maintain links to other Aboutcode Projects.

To link sections in the same documentation, standart reST labels are used. Refer Cross-Referencing for more information.

For example:

```
.. _my-reference-label:

Section to cross-reference
--------------------------

This is the text of the section.

It refers to the section itself, see :ref:`my-reference-label`.
```

Now, using Intersphinx, you can create these labels in one Sphinx Documentation and then referance these labels from another Sphinx Documentation, hosted in different locations.

You just have to add the following in the `conf.py` file for your Sphinx Documentation, where you want to add the links:

```
extensions = [
'sphinx.ext.intersphinx'
]

intersphinx_mapping = {'scancode-toolkit': ('https://scancode-toolkit.readthedocs.io/en/
→latest/', None)}
```

To show all Intersphinx links and their targets of an Intersphinx mapping file, run:

```
python -msphinx.ext.intersphinx https://scancode-toolkit.readthedocs.io/en/latest/
→objects.inv
```

> **Warning:** `python -msphinx.ext.intersphinx https://scancode-toolkit.readthedocs.io/objects.inv` will give error.

This enables you to create links to the `scancode-toolkit` Documentation in your own Documentation, where you modified the configuration file. Links can be added like this:

```
For more details refer :ref:`scancode-toolkit:doc_style_guide`.
```

You can also not use the `scancode-toolkit` label assigned to all links from scancode-toolkit.readthedocs.io, if you don't have a label having the same name in your Sphinx Documentation. Example:

```
For more details refer :ref:`doc_style_guide`.
```

If you have a label in your documentation which is also present in the documentation linked by Intersphinx, and you link to that label, it will create a link to the local label.

For more information, refer this tutorial named Using Intersphinx.

2.4.8 Extra Style Checks

1. Headings

   (Refer) Normally, there are no heading levels assigned to certain characters as the structure is determined from the succession of headings. However, this convention is used in Python's Style Guide for documenting which you may follow:

   # with overline, for parts

   - with overline, for chapters

   =, for sections

   -, for subsections

   ^, for sub-subsections

   ", for paragraphs

2. Heading Underlines

   Do not use underlines that are longer/shorter than the title headline itself. As in:

```
Correct :

Extra Style Checks
------------------

Incorrect :

Extra Style Checks
-----------------------
```

Note: Underlines shorter than the Title text generates Errors on sphinx-build.

3. Internal Links

   Using `:ref:` is advised over standard reStructuredText links to sections (like `` `Section title`_ ``) because it works across files, when section headings are changed, will raise warnings if incorrect, and works for all builders that support cross-references. However, external links are created by using the standard `` `Section title`_ `` method.

4. Eliminate Redundancy

   If a section/file has to be repeated somewhere else, do not write the exact same section/file twice. Use `.. include:  ../README.rst` instead. Here, `../` refers to the documentation root, so file location can be used accordingly. This enables us to link documents from other upstream folders.

5. Using `:ref:` only when necessary

   Use `:ref:` to create internal links only when needed, i.e. it is referenced somewhere. Do not create references for all the sections and then only reference some of them, because this created unnecessary references. This also generates ERROR in `restructuredtext-lint`.

6. Spelling

   You should check for spelling errors before you push changes. Aspell is a GNU project Command Line tool you can use for this purpose. Download and install Aspell, then execute `aspell check <file-name>` for all the files changed. Be careful about not changing commands or other stuff as Aspell gives prompts for a lot of them. Also delete the temporary `.bak` files generated. Refer the manual for more information on how to use.

7. Notes and Warning Snippets

   Every `Note` and `Warning` sections are to be kept in `rst_snippets/note_snippets/` and `rst_snippets/warning_snippets/` and then included to eliminate redundancy, as these are frequently used in multiple files.

2.4.9 Converting from Markdown

If you want to convert a `.md` file to a `.rst` file, this tool does it pretty well. You'd still have to clean up and check for errors as this contains a lot of bugs. But this is definitely better than converting everything by yourself.

This will be helpful in converting GitHub wiki's (Markdown Files) to reStructuredtext files for Sphinx/ReadTheDocs hosting.

OTHERS

3.1 AboutCode Data Structure (ABCD)

3.1.1 Summary

ABCD is an abbreviation for ABout Code Data. The AboutCode Data goal is to provide a simple, standardized and extensible way to document data about software code such that:

- It is a common way to exchange data about code between any nexB tools by import and export.

- It becomes the preferred way to exchange data between nexB tools and other tools.

- It could become a valuable structure to exchange data between any tools concerned with data about software.

ABCD is technology and programming language neutral, preferring JSON or YAML document formats.

ABC Data is structured around a few basic objects: Products, Components, Packages, Files, Parties and Licenses. It is extensible to other specific or future object types.

Objects have "attributes" that are simple name/value pairs. A value can be either a plain value or another object or a list of objects and attributes.

ABC Data is minimally specified by design: only a few basic objects and attributes are documented with conventions to name and structure data and how to define relationships between objects. There is only a small reference dictionary for some well known attributes documented here.

The planned benefit for tools using ABC Data is simplified data exchange and integration between multiple best-of-breed tools.

3.1.2 Context

There is currently no easy way to describe information about code and software in a simple and standardized way. There have been many efforts to provide this data in a more structured way such as:

- SPDX (focused on packages and licenses),

- DOAP (focused on projects),

- The original ABOUT metafile format, and

- The many different software package metadata formats (Maven, NPM, RPM, Deb, etc).

These data structures are fragmented and generally too purpose- or technology-specific.

Recently there have been efforts to collect and expose more data such as:

- libraries.io (a catalog of packages, AGPL-licensed) and dependencyci.com its companion commercial service,

- versioneye.com (a catalog of package versions updates, now MIT-licensed),

- softwarearchive.org (an effort to build an all-encompassing software source code archive),

- sources.debian.net (a Debian-focused code and metadata search facility),

- searchcode.com (an add-supported source code search engine exposing some metadata)

- appstream (a cross-distro effort to normalize desktop package metadata access to Linux desktops https://www.freedesktop.org/software/appstream/docs/).

These efforts are all useful, but they do not address how you can consistently exchange data about code in a user-centric and technology-neutral, normalized way.

Why does this matter? Software and code are everywhere. FLOSS code is exploding with millions of components and packages. The data about this code is out there somewhere but getting it is often too difficult. This is a problem of data normalization, aggregation and exchange.

Whether you consume or produce software, accessing and creating normalized data about your code and the code you use should be made easier such that:

- You can efficiently make code selection and re-use decisions,

- You can discover what is in your code, and

- You can continuously track updates, bugs, licenses, liveliness, quality and security attributes of the code you use or consider using.

With the notable exceptions of SPDX and the earlier versions of the ABOUT format, available data formats about software have been designed first for a specific technology (e.g. Linux distros) or programming language (e.g. maven, npm, etc.) and documentation of code provenance and related attributes has been secondary and minimal. In most cases, the primary focus has been to provide first comprehensive support for package installation, dependency resolution or package building and provenance and licensing information is often treated with lesser details.

3.1.3 ABCD: the AboutCode Data structure

ABCD is an abbreviation for ABout Code Data. The goal is to provide a simple, standardized and extensible way to document things about software code.

In contrast with other approaches, the AboutCode Data structure is focused on providing data that is useful to users first and is not limited to software package data only. AboutCode Data need not be as strictly specified as traditional package manager data formats because its purpose is not to drive a software build, package creation or software installation nor is it to compute the resolution of dependencies. It only provides information (metadata) about the code.

The vision for the ABC Data structure is to provide a common way to exchange data about code between all nexB tools, such that these tools can all import and export data about code seamlessly (TraceCode, ScanCode, AboutCode Manager, AttributeCode, upcoming MineCode, etc.). The ABCD structure should also be the preferred way to exchange data about code between nexB tools and other tools. We may create small adapters to convert other data formats in and out of the ABCD structure and encourage other tool authors to natively support ABC Data, though the main focus is on our tools.

The ABCD structure is technology and programming language neutral and designed so that the parties exchanging data about code can do so reliably with some minimal conventions; and that the data is easily processed by machines and not hard to read by humans.

ABC Data is structured around "objects". Objects have "attributes" that are simple name/value pairs. A value can be either a plain value or another object or a list of nested objects and attributes.

ABC Data is organized around:

- a few known object types,

- simple conventions to create lists of objects and describe object relationships,

- simple conventions to create attributes as name/value pairs, and

- a small dictionary or vocabulary of well-known attribute names that have a common definition across all tools.

ABC Data is "under-specified" by design: only a few essential objects and attributes are documented here with the conventions on how to structure the ABC Data.

3.1.4 Basic objects describing data about code

At the top level we have these main object types:

- Product(s): a software product, application or system such as a Cost Accounting application.

- Component(s): a software component, such as the PostgreSQL 9 database system, usually a significant or major version

- Package(s): a set of files that comprise a Component as used, such as a postgresql-9.4.5-linux-x86.zip archive. The version is exact and specific.

- File(s): any file and directory identified by a path, such as a binary package or a source code directory or file.

And these secondary, important but less prominent object types:

- Party(ies): a person or an organization. An organization can be a project, formally or informally organized, a company, a department within a company, etc. A Party typically has contact information (such as an email or physical address or home url). A Party may have defaults that apply to much of its software (for an org that creates software) such as a default Apache license for Apache Foundation projects. Parties often relate to other objects through a role relationship such as owner, author, maintainer, etc.

- License(s): information about the license of code. A License typically has a name, text and additional categories. (tags or attributes).

Each of these objects has a few identifying attributes and eventually many tool- or application-specific data attributes. Each tool defines and documents the attributes they can handle and care for. When some agreement is reached on the definition of new attributes or objects, the ABCD dictionary may be updated accordingly with new objects types such as for software security, quality or other interesting aspects.

Objects are interrelated with other objects. Objects can relate to each other via a reference using identifiers pointing to other objects or via an embedded list of objects. The nature of the relationship between two objects can also be specified with additional attributes as needed.

Here are typical relationships between objects:
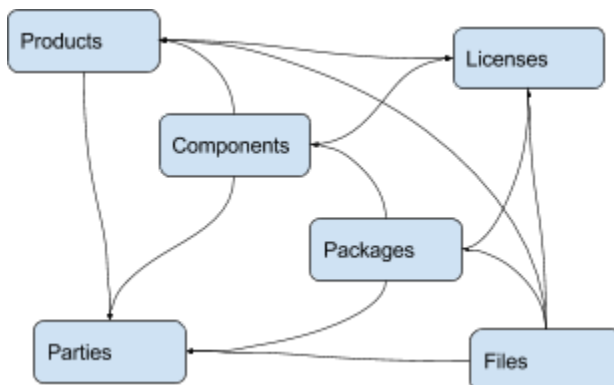
Here is an example of relationships for a simple Widget product:
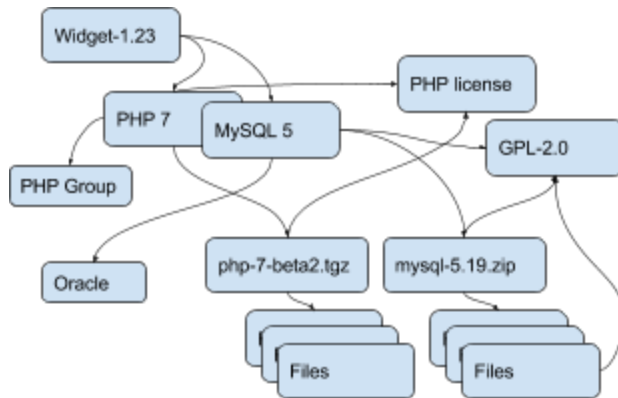
Tools can define any custom objects and some used more commonly may be promoted to be documented here over time.

3.1.5 Attribute Names and Values

By convention, a tool receiving ABC Data should process only the data it knows and should ignore unknown attributes or objects. This is important to allow the data structure to evolve and provide some forward and backward compatibility. When an ABCD payload contains data elements that a receiver does not know about, the receiver should still be able to process the known objects and attributes.

- Attributes are name/value pairs.

- Attribute names are always strings, not numbers, not booleans, not any other data format. In these strings, leading and trailing white spaces (spaces, tabs, line returns, etc) are not significant and can be safely ignored or removed.

- Attribute values are one of the standard JSON types: string, number, boolean or null. In strings, leading and trailing white spaces (spaces, tabs, line returns, etc) are not significant and can be safely ignored or removed.

- Self-explicit names should be used rather than obscure names or abbreviations: names should be self-explicit and self-evident.

Except for the data organization conventions described here and the use of the well-known object and attribute names, nothing is mandatory in the ABCD format. This means that even partial, incomplete or sketchy data about code can be transferred in this format.

The meaning of well known object names such as Product, Component, Package, File, Party and License is defined in this document.

3.1.6 Name conventions

- Names are strings composed ONLY of ASCII letters, numbers or underscores. Names cannot start with a number. Names cannot contain spaces nor other punctuation, not even a dot or period.

- Names are NOT case sensitive: upper or lowercase does not matter and the standard is to use lowercase. It is a mistake to use upper or mixed case but this is something a parser receiving ABC Data should recover from nicely by converting the names to lowercase.

- Names are made of English words: there is no provision currently for non-English names. Tools that deal with multilingual content may define their own conventions to provide content in other languages. ABCD may add one of these conventions in the future.

- Parser implementation can be smarter and gentler: For names, anything that is not ASCII or number or underscore can be accepted by a parser and could be replaced by an underscore, including a starting digit if any. Or a parser may provide a warning if there is an unknown name that is very close to a well known name. Or a parser may

accept CamelCase and transform names to underscore_case and perform another transformation to conventional ABC Data.

- Names are singular or plural: When a name refers to more than one item, the name of the field is plural and the value is a list of values. For instance "url" and "urls".

- Top level known objects are ALWAYS plural and stored in lists: "parties" or "files" or "products" or "components". This makes it easier to write tools because the top level types are always lists, even when there is a single object in that list.

- A value must not be used as a name: in an attribute name/value pair, the name is always a name, not a value and every value must have a name.

- For instance, this JSON snippet would not be correct where a URL is used as a name:

```
{"http://someurl.com": "this is the home URL"}
```

- Use rather this form to specify a name for the URL attribute:

```
{"url": "http://someurl.com", "note": "this is the home URL"}
```

- But this would be correct when using a list of plain values where "urls" is plural:

```
{"urls": ["http://someurl.com", "http://someurl2.com"]}
```

- An attribute names without a value is not needed. Only names with values are needed, and attributes without values can be omitted: each tool may do what it wants for these cases. For instance it may be handy to provide all attributes even if not defined in an API payload. But when serializing data as YAML, meant for human editing, including all empty values may not help with reading and processing the YAML text. An undefined attribute without a set value should be assigned with the null JSON value: this has the same meaning as if the attribute was not specified and absent from the payload. If you want to specify that an attribute has an empty value and does not have a value (as opposed to have an unknown value) use an empty string instead.

- Avoid abbreviated names, with some exceptions. Names should always be fully spelled out except for:

  - url: uniform resource locator

  - uri: uniform resource identifier

  - urn: uniform resource name

  - vcs: version control system

  - uuid: universally unique identifier, used for uuid4 string https://tools.ietf.org/html/rfc4122.html

  - id: identifier

  - info: information

  - os: operating system

  - arch: architecture

- For some common names we use the common compound form such as:

  - codebase: and not code_base

  - filename: and not file_name

  - homepage: and not home_page

Well known attribute names include:

- name: the name of a product, component, license or package.

- version: the version of a product, component, package.

- description: description text.

- type: some type information about an object. For instance, a File type could be: directory, file or link.

- keywords: a list of keywords about an object. For example, the keywords of a component used to "tag" a component.

- path: the value is the path to a file or directory, either absolute or relative and using the POSIX convention (a forward slash as separator). For Windows paths, replace backslash with forward slashes. Directories should end with a slash in a canonical form.

- key: the value is some key string, slug-like, case-insensitive and composed only of ASCII letters and digits, dash, dot and underscore. No white spaces. For example: org.apache.maven-parent

- role: the value describes the role of a Party in a relationship with other objects. For instance a Party may be the "owner" or "author" of a Component or Package.

- uuid: a uuid4 string https://tools.ietf.org/html/rfc4122.html

- algorithms for checksums: to store checksums we use a name/value pairs where the name is an algorithm such as sha1 and the value is a checksum in hexadecimal such as "sha1": "asasa231212" . The value is the standard/default string created by command line tools such as sha1sum. Supported algorithms may evolve over time. Common checksums include md5, sha1, sha256, sha512.

- notes: some text notes. This is an exception to the singular/plural rule for names: notes is a single text field and not a list.

As the usage of the ABCD structure matures, more well known names will be documented in a vocabulary.

3.1.7 Value conventions

- Attribute values are one of the standard JSON types: string, number, boolean or null. In strings, leading and trailing white spaces (spaces, tabs, line returns, etc) are not significant and can be safely ignored or removed.

- To represent a date/time use the ISO format such as 2016-08-15 defaulting to UTC time zone if the time zone is not specified in the date/time stamp.

- All string values are UTF-8 encoded.

Well known name prefixes or suffixes can be used to provide a type hint for the value type or meaning:

- xxx_count, xxx_number, xxx_level: the value is an integer number. Example: results_count or curation_level

- date_xxx or xxx_date: the value is a date/time stamp in ISO format such as 2016-08-16 (See https://www.ietf.org/rfc/rfc3339.txt). Examples: last_modified_date, date_created

- xxx_url: the value is a URL for web http(s) or ftp url that points to an existing valid web resource (that could possibly no longer exist on the web). Example: homepage_url or api_url

- xxx_uri: the value is a URI typically used as an identifier that may not point to an existing web resource. Example: git://github.com/nexb/scancode-toolkit

- xxx_file or xxx_path: the value is a file path. This can come handy for external files such as a license file. Example: notice_file

- xxx_filename: the value is a file name. Example: notice_filename

- xxx_text: the value is a long text. This is only a hint that it may be large and may span multiple lines. Example: notice_text

- xxx_line: such as start_line and end_line: the value is a line number. The first line number is 1.

- xxx_status: such as configuration_status. Indicates that the value is about some status.

- xxx_name: such as short_name. Indicates that the value is a name. Commonly used for long_name, short_name. The bare name shout be preferred for the obvious and most common way an object is named.

- xxx_flag, is_xxx, has_xxx: such as is_license_notice. Indicates that the string value is a boolean.

3.1.8 Object identifiers

We like objects to be identifiable. There is a natural way to identify and name most objects: for instance, the full name of a person or organization or the name and version of a Component or Package or the path to a File, are all natural identifiers to an object.

However, natural names are not always enough to fully identify an object and may need extra context to reference an object unambiguously. There could be several persons or organizations with the same name at a different address. Or the foo-1.4 Package could be available as a public RubyGem and also as an NPM; or a private Python package foo-1.4 has been created by a company and is also available on Pypi. Or the "foo" Package is the name of a Linux Package, an NPM and a Ruby Package but these three packages are for unrelated components.

Hence each object may need several attributes to be fully identifiable.

For example, public package managers ensure that a name is unique within the confines of a source. "logging" is the unique name of a single Sourceforge project at https://sourceforge.net/projects/logging/. "logging" is the unique name of an Apache project at the Apache Foundation http://logging.apache.org/.

Yet, these two names point to completely different software. In most cases, providing information about the "source" where an identifier is guaranteed to be unique is enough to ensure proper identification. This "source" is easily identified by its internet source name, and an internet source name is guaranteed to be unique globally. The "source" of identifiers is not mandatory but it is strongly encouraged to use as an attribute to provide good unique identifiers. Still, tools exchanging ABC Data must be able to exchange under-specified and partially identified data and may sometimes rely on comparing many attributes of two objects to decide if they are the same.

The minimal way to identify top level objects is the combination of a "source" and a unique identifier within this source. The source can be implicit when two parties are exchanging data privately or explicit using the "source" attribute.

Within a source, we use the most obvious and natural identifies for an object. For example:

- For Products, Components and Packages we can use their name and version.

- For Files we use a path of a file or directory, possibly relative to a package or a product codebase; or a checksum of a file or archive such as a sha1.

- For Parties, we use a name possibly supplemented with a URL or email.

- For all object types we can use a "universally unique id" or UUID-4 (https://tools.ietf.org/html/rfc4122.html)

- For all object types, we can use a key, which is a slug-like string identifier such as a license key.

- For all object types, we can use a URN (https://en.wikipedia.org/wiki/Uniform_resource_name) Tools may also define their own URNs, namespaces and names such as a DejaCode urn is, urn:dje:component:16fusb:1.0

Beyond direct identification, an object may have several alternative identifiers, aka "external references". For instance a Package may have different names and slightly different versions in the Linux, Debian or Fedora distros and a Pypi Package with yet another name where all these Packages are for the same Component and the same code. Or a Party such as the Eclipse Foundation may be named differently in DejaCode and the NVD CPEs.

To support these cases, the "external_reference(s)" attribute can be used where needed in any object to reference one or more external identifiers and what is the source for this identifier (note: "external" is really a matter of point of view of who owns or produces the ABC Data). An attribute with name suffix of "xxx_reference" may also be used to provide a simpler external reference, such as "approval_reference".

For example, this ABC Data could describe the external id of Party to a CPE and to TechnoPedia (here in a YAML format):

```yaml
parties:
  - name: Apache Foundation
    homepage_url: http://apache.org
    type: organization
    external_references:
        - source: nvd.nist.gov
          identifier: apache
        - source: technopedia.com
          identifier: Apache Foundation (The)
        - source: googlecode.com
          identifier: apache-foundation
```

Other identifiers may also be used, as needed by some tools, such as in hyperlinked APIs.

3.1.9 Organizing data and relationships

Describing relationships between objects is essential in AboutCode Data. There are two ways to describe these relationships: by referencing or by embedding objects.

When using a reference, you relate objects by providing identifiers to these objects and may provide additional object details in separate lists. When embedding, you include not only the reference but also the related object details in another object data. This could include all data about an object or a subset as needed.

For example, this components list embeds a list of two packages.

Note: "components" is always a list, *even when it has a single component*:

```json
{"components": [{
    "source": "http://apache.org",
    "name": "Apache httpd",
    "version": "2.3",
    "packages": [
        {"name": "httpd",
         "version": "2.3.4",
         "download_url": "http://apache.org/dist/httpd/httpd-2.3.4.zip",
         "sha1": "acbf23256361abcdf",
         "size": 3267,
         "filename": "httpd-2.3.4.zip"
        },

        {"name": "httpd",
         "version": "2.3.5",
         "download_url": "http://apache.org/dist/httpd/httpd-2.3.5.tar.gz",
         "sha1": "ac8823256361adfcdf",
         "size": 33267,
         "filename": "httpd-2.3.5.tar.gz"
        }
    ]
}]}
```

In this example, the component list references two packages that are listed separately and uses the checksum as package identifiers for the reference. This data is strictly equivalent to the previous example but using a different layout. When

all the data is provided, the effect of embedding or referencing objects results in the same data, just organized differently:

```json
{"components": [{
    "source": "http://apache.org",
    "name": "Apache httpd",
    "version": "2.3",
    "packages": [
        {"sha1": "aacbf23256361abcdf"},
        {"sha1": "ac8823256361adfcdf"}
    ]
}],

"packages": [
    {"name": "httpd", "version": "2.3.4",
     "download_url":
     "http://apache.org/dist/httpd/httpd-2.3.4.zip",
     "sha1": "acbf23256361abcdf", "size": 23267, "filename": "httpd-2.3.4.zip"},

    {"name": "httpd", "version": "2.3.5",
     "download_url": "http://apache.org/dist/httpd/httpd-2.3.5.tar.gz",
     "sha1": "ac8823256361adfcdf", "size": 33267, "filename": "httpd-2.3.5.tar.gz"}
]}
```

In this third example the packages are referencing one component instead. That component is always wrapped in a components list. The component detail data is not provided. The details may be available elsewhere in a tool that tracks components:

```json
"packages": [
    {"name": "httpd", "version": "2.3.4",
     "download_url": "http://apache.org/dist/httpd/httpd-2.3.4.zip",
     "sha1": "acbf23256361abcdf", "size": 23267, "filename": "httpd-2.3.4.zip",
     "components": [
        {"source": "http://apache.org", "name": "Apache httpd", "version": "2.3"}
     ]
    },

    {"name": "httpd", "version": "2.3.5",
     "download_url":"http://apache.org/dist/httpd/httpd-2.3.5.tar.gz",
     "sha1": "ac8823256361adfcdf", "size": 33267, "filename": "httpd-2.3.5.tar.gz",
     "components": [
        {"source": "http://apache.org", "name": "Apache httpd", "version": "2.3"}
     ]
    }
]
```

Relationships can be documented with this approach in different ways. Typically when the primary concern is about a Product, then the Product object may embed data about its Components. When the primary concern is Packages, they may embed or reference Products or Components or files. For example:

- A tool may prefer to provide data with products or components as top level objects. The components used in a Product are naturally embedded in the products.

- A tool concerned more with files, will provide files as top level objects and may embed package details when they are found for a file or directory path.

- Another tool may focus on packages and provide packages first with component references and possibly embed-

ded files. A matching tool may provide packages first and reference matched files. The file paths of a package are naturally embedded in the package, though using references may help keep the data simpler when there is a large volume of files.

- A tool that generates attribution documentation may focus first on components and second on licenses or packages references.

- A tool dealing with security vulnerabilities may define a Vulnerability object and reference Packages and Files that are affected by a Vulnerability.

To better understand the embedding or referencing relationships:

- using references is similar to a tabular data layout, akin to a relational database table structure

- using embedding is similar to a tree data layout such as in a file/directory tree or nested data such as XML.

Another way to think about these relationships is a "GROUP BY" statement in SQL. The data can be grouped-by Component, then Packages or grouped-by Files then Components.

Both referencing and embedding layouts can be combined freely and are not mutually exclusive. When using both at the same time, some care is needed to avoid creating documents with conflicting or duplicated data that is referenced and embedded at the same time.

Using references is often useful when there is an agreement on how to reference objects between two tools or parties. For instance, when using nexB tools, a unique and well defined license key is used to reference a license rather than embedding the full license details. A concise reference to the name and version of a public package from a well known package repository such as RPM or Maven can be used to the same effect. Or an SPDX license identifier can be used to reference an SPDX-listed license without having to embed its full license text.

The nature of the relationship between two objects can be specified when it is not obvious and requires some extra specification. Each tool can define additional attributes to document these. For instance a common relationship between a party and a product or component is a role such as owner. For packages a role can be maintainer, author, etc. Or the license of a file or package may be the "asserted" license by the project authors. It may differ from the "detected" license from a scan or code inspection and may further differ from a "concluded" license or a "selected" license when there is a license choice. At the package and license level the types of relationships documented in the SPDX specification are a good source for more details. For example this component references two parties where one is the author and the other is the maintainer documented using a role attribute:

```
"components": [{
    "source": "http://apache.org",
    "name": "Apache httpd",
    "version": "2.3",
    "parties": [
        {"name": "John Doe", "type": "person", "role": "author"},
        {"name": "Jane Smith", "type": "person", "role": "maintainer"},
        {"name": "Jane Smith", "type": "person", "role": "owner"},
    ]
}]
```

3.1.10 Document format conventions

The default ABC Data format is JSON (though it can be serialized to anything else that would preserve its structure). YAML is also supported and preferred for storage of simple documents that document one or a few top level objects and that need to be edited by a human.

The data structure by nested name/value pairs attributes and lists of values maps naturally to the corresponding JSON and YAML constructs. In JSON-speak these are arrays (lists) and objects (name/value pairs).

ABC Data can be provided as simple files or embedded in some API payload. As files, their content can be either JSON or YAML and should have either a .json or .yml extension by convention. For backwards compatibility with previous AboutCode conventions, the .ABOUT extension can be used for YAML documents. For instance this is used in the legacy about_code_tool and its successors. The DocumentCode tool can store individual attribution data in a .ABOUT yml file.

The top level structure of an ABC Data block is always a JSON object or YAML dictionary. Depending on the context this top level structure may be wrapped in another data structure (for instance when exchanging AboutCode Data in some web api, the API may provide ABC Data as a payload in a "results" or "body" or "data" block and also have some "headers" or "meta" block).

The top level elements must contain at least one of the object names and a list of objects such as here with a list of files:

```
files:
    - path: this/foo/bar
      size: 123
      sha1: aaf35463472abcd
    - path: that/baz
```

Optionally an "aboutcode_version" attribute can be added at the top level to document which version of the AboutCode Data structure is used for a document. For example: aboutcode_version: 4.0

Order of attributes matters to help reading documents: tools that write ABC Data should attempt to use a consistent order for objects and attribute names rather than a random ordering. However, some tools may not be able to set a specific order so thi is only a recommendation. The preferred order is to start with identifiers and keys and from the most important to the least important attributes, followed by attributes grouped logically together, followed by related objects.

3.1.11 References between documents and payload, embedding other files

ABC Data may reference other data. For instance in a hyperlinked REST API a list of URLs to further invoke the API and get license' details may be provided with an api_url attribute to identify which API calls to invoke. The ways to reference data and the semantics and mechanics of each of these embeddings or references needed to get the actual data are not specified here. Each tool may offer its own mechanism. A convention for an hyperlinked REST API JSON payload could be to use api_url(s) identifier to specify additional "GET"able endpoints. The AttributeCode tool use *_file attributes in YAML or JSON documents to reference external license and notices text files to load with the text content.

Another convention is used in ScanCode to reference license texts and license detection rules by key: An ABC Data YAML file contains the ABC Data. And side by side there is a file with the same base name and a LICENSE, SPDX or NOTICE, RULE, extension that contains the actual text corresponding to the license, the SPDX text or the rule text. The convention here is to use an implicit reference between files because they have the same base name and different extensions.

In the future, we may specify how to embed an external ABC Data file in another ABC Data file; this would only apply to file-based ABC Data payload though and could not apply to hyperlinked REST APIs.

3.1.12 Document-as-files naming, exchange and storage

Each tool handling ABC Data may name an ABC Data file in any manner and store the data in any way that is appropriate. The structure is a set of data exchange conventions and may be used for storage but nothing is specified on how to do this.

For consistency, tools consuming AboutCode Data are encouraged to use the same data structure internally and in their user interface to organize and name the data, but this is only a recommendation.

For instance, the AttributeCode tool uses a convention to store ABC Data as YAML in a file with a .ABOUT extension and uses the ABC Data structures internally and externally.

When exchanging data (for instance over an API), the API provider of ABC Data should support a request to return either embedded data or data by reference and ideally allow the caller to specify which objects and attributes it is interested in (possibly in the future using something like GraphQL).

When interacting with tools through an API, the conversation could start by sending an ABC Data payload with some extra request data and receiving an ABC Data payload in return. For instance, when requesting matching packages from a matching tool, you could start by passing scan data with checksums for several files at once and receive detailed data for each of the matched files or packages.

3.1.13 Documenting and validating attributes

Each tool handling ABC Data may only be interested in processing certain objects and attributes when accepting data in, or when providing data out. Attributes that are unknown should be ignored. To document which objects and which attributes a tool can handle, a tool should provide some documentation. The documentation format is not specified here, but it could use a JSON schema in the future. This should include documentation regarding if and how data is validated, and when and how errors or warnings are triggered and provided when there is a validation error. For example, a validation could be to check that an SPDX license id exists at SPDX or that a URL is valid.

3.1.14 Notes on YAML format

YAML is the preferred file format for ABC Data destined for reading or writing primarily by humans.

- Block-style is better.

- When you write AboutCode Data as YAML, you should privilege block-style and avoid flow-style YAML which is less readable for humans.

- Avoid Multi-document YAML.

- Multi-document YAML documents should be avoided (when using the — separators).

- Beware of parser shenanigans: Most YAML parsers recognize and convert automatically certain data types such as numbers, booleans or dates. You should be aware of this because the ABC Data strings may contain date stamps. You may want to configure a YAML parser to deactivate some of these automated format conversions to avoid unwanted conversions.

3.1.15 Notes on JSON Format

JSON is the preferred file format for ABC Data destined for reading and writing primarily by machines.

- "Streamable" JSON with JSON-lines.

A large JSON document may benefit from being readable line-by-line rather than loaded all at once in memory. For this purpose, the convention is to use JSON lines where each line in the document is a valid JSON document itself: this enables reading the document in line-by-line increments. The preferred way to do so is to provide one ABCD top level object per document where the first line contains meta information about the stream such as a notice, a tool version or the aboutcode version.

- Avoid escaped slash.

The JSON specification says you CAN escape forward slash, but this is optional. It is best to avoid escaping slash when not needed for better readability.

For instance for URLs this form:

```
"https://enterprise.dejacode.com/component_catalog/nexB/16fusb/1.0/"
```

should be preferred over this escaped form when backslashes are not needed:

```
"https:\\/\\/enterprise.dejacode.com\\/component_catalog\\/nexB\\/16fusb\\/1.0\\/"
```

3.1.16 Notes on embedding ABC Data in source code files.

It could be useful to include ABC Data directly in a source code file, such as to provide structured license and provenance data for a single file. This requires of course a file modification. While this is not a preferred use case, it can be handy to document your own code one file at a time. Using an external ABC Data file should be preferred but here are conventions for this use case:

- The ABC Data should be embedded in a top level block of comments.
- Inside that block of comments the preferred format is YAML.
- How a tool collects that ABC Data when embedded in code is to be determined.
- Tools offering such support should document and eventually enforce their own conventions.

3.1.17 Notes on spreadsheet and CSV files

ABC Data does not support or endorse using CSV or spreadsheets for data exchange.

CSV and other spreadsheet file formats are NOT recommended to store ABC Data. In most cases you cannot store a correct data set in a spreadsheet. However, these tools are also widely used and convenient. Here are some recommendations when you need to communicate ABC data in a CSV or spreadsheet format: even though ABC Data is naturally nested and tree-like, it should be possible to serialize certain ABCD objects as flat, tabular data.

- Naming columns

The table column names may need to be adjusted to correctly reference the multiple level of object and attribute nesting using a dot as a separator. The dot or period is otherwise not allowed in attribute names. For example, you could use files.path for files or components.name to reference a component name. Some tools may prefer to create tabular files with their own column names and layout, and provide mappings to ABC Data attribute and object names.

- Example for an inventory:

Since ABC Data can be related by reference, the preferred (and cumbersome) way to store ABC Data in a spreadsheet is to use one tab for each object type and use identifying attributes to relate objects between each others across tabs. For instance, in a Bill of Materials (BOM) spreadsheet for a Product, you could use a tab to describe the Product attributes and another tab to describe the Components used in this Product and possibly additional tabs to describe the related packages and files corresponding to these

- Care is needed for Packages, Components and other names and for dates, versions, unicode and UTF-8 to avoid damaging content (aka. mojibake)

Spreadsheet tools such as Excel or LibreOffice automatically recognize and convert data to their own format: a date of 20016-08-17 may be converted to a date number when a CSV is loaded and difficult to recover as a correct original date stamp string afterwards. Or a version 1.0 may be irreversibly converted to 1 or 1.90 to 1.9 losing important version information.

Spreadsheet tools may not recognize and handle properly UTF-8 texts and damage descriptions and texts. These tools may also treat strings starting with the equal sign as a formula. When incorrectly recognizing special accentuated characters this may damage texts creating what is called "mojibake" (See https://en.wikipedia.org/wiki/Mojibake)

Always use these tools with caution and be prepared for damage to your data if you use these tools to save or create ABC Data.

Impact on AttributeCode

As an integration tool, AttributeCode itself may specify only a very few elements.

The new structure will need to be implemented. Here could be an example in YAML:

```yaml
aboutcode_version: 4.0
components:
-   source: dejacode.com
    name: bitarray
    version: 0.8.1
    homepage_url: https://github.com/ilanschnell/bitarray
    copyright: Copyright (c) Ilan Schnell and others
    files:
        - path: some/directory/
          type: dir
        - path: bitarray-0.8.1-cp27-cp27m-macosx_10_9_intel.whl
        - path: someotherdir/bitarray-0.8.1-cp27-cp27m-manylinux1_i686.whl
        - path: bitarray-0.8.1-cp27-cp27m-manylinux1_x86_64.whl
        - path: bitarray-0.8.1-cp27-cp27m-win_amd64.whl
        - path: bitarray-0.8.1-cp27-cp27m-win32.whl
        - path: bitarray-0.8.1-cp27-cp27mu-manylinux1_i686.whl
        - path: bitarray-0.8.1-cp27-cp27mu-manylinux1_x86_64.whl
        - path: bitarray-0.8.1-cp27-none-macosx_10_6_intel.whl
        - path: bitarray-0.8.1.tar.gz

    parties:
      - role: owner
        name: Ilan Schnell

    packages:
      - download_url: http://pypi.python.org/packages/source/b/bitarray/bitarray-0.8.1.
→tar.gz
        sha1: 468456384529abcdef342
```

(continues on next page)

```
    license_expression: psf

    licenses:
      - source: scancode.com
        key: psf
        text_file: PSF.LICENSE
```

And here would be similar data in JSON:

```json
{"components": [{
              "name": "bitarray",
              "version": "0.8.1"
              "homepage_url": "https://github.com/ilanschnell/bitarray",
              "copyright": "Copyright (c) Ilan Schnell and others",
              "license_expression": "psf",
              "licenses": [{"key": "psf", "text_file": "PSF.LICENSE", "source":
→"scancode.com"}],
              "packages": [{"download_url": "http://pypi.python.org/packages/source/b/
→bitarray/bitarray-0.8.1.tar.gz"
                           "sha1": "468456384529abcdef342"
              }],
              "parties": [{"name": "Ilan Schnell", "role": "owner"}],

              "files": [{"path": "some/directory/", "type": "dir"},
                       {"path": "bitarray-0.8.1-cp27-cp27m-macosx_10_9_intel.whl"},
                       {"path": "bitarray-0.8.1-cp27-cp27m-manylinux1_i686.whl"},
                       {"path": "bitarray-0.8.1-cp27-cp27m-manylinux1_x86_64.whl"},
                       {"path": "bitarray-0.8.1-cp27-cp27m-win_amd64.whl"},
                       {"path": "bitarray-0.8.1-cp27-cp27m-win32.whl"},
                       {"path": "bitarray-0.8.1-cp27-cp27mu-manylinux1_i686.whl"},
                       {"path": "bitarray-0.8.1-cp27-cp27mu-manylinux1_x86_64.whl"},
                       {"path": "bitarray-0.8.1-cp27-none-macosx_10_6_intel.whl"},
                       {"path": "bitarray-0.8.1.tar.gz"}],
              }],
 aboutcode_version: "4.0"}
```

Impact on ScanCode Toolkit

The new format will need to be implemented for scan results in general and for packages in particular.

ScanCode will specify Package and several attributes related to scanning and referencing clues for files, directories and packages.

Alternatively Packages could be extracted to an independent PackagedCode library.

The changes will minimize impact on the layout of the scan results. Here is an example of a scan payload in ABCD format: this is essentially the standard scan format:

```json
{
  "scancode_notice": "Generated with ScanCode and provided .......",
  "scancode_version": "2.0.0.dev0",
```

```
  "files_count": 7,
  "files": [
    {
      "path": "samples/JGroups/src/",
      "type": "directory",
      "files_count": 29
      "licenses" : [
          { "key":"apache-2.0",
            "concluded": true}
      ]
    }
    {
      "path": "samples/JGroups/src/GuardedBy.java",
      "date": "2015-12-10",
      "programming_language": "Java",
      "sha1": "981d67087e65e9a44957c026d4b10817cf77d966",
      "name": "GuardedBy.java",
      "extension": ".java",
      "file_type": "ASCII text",
      "is_text": true,
      "is_source": true,
      "md5": "c5064400f759d3e81771005051d17dc1",
      "type": "file",
      "is_archive": null,
      "mime_type": "text/plain",
      "size": 813,
      "copyrights": [
        {
          "end_line": 12,
          "start_line": 9,
          "holder": "Brian Goetz and Tim Peierls",
          "statement": "Copyright (c) 2005 Brian Goetz and Tim Peierls"
        }
      ],
      "licenses": [
        { "detected": true,
          "key": "cc-by-2.5",
          "short_name": "CC-BY-2.5",
          "homepage_url": "http://creativecommons.org/licenses/by/2.5/",
          "dejacode_url": "https://enterprise.dejacode.com/license_library/Demo/cc-by-2.
→5/",
          "text_url": "http://creativecommons.org/licenses/by/2.5/legalcode",
          "owner": {
            "name": "Creative Commons",
          },
          "detection_score": 100.0,
          "start_line": 11,
          "end_line": 11,
          "category": "Attribution",
          "external_reference": {
            "source": "spdx.org",
            "key": "CC-BY-2.5"
```

```
          "url": "http://spdx.org/licenses/CC-BY-2.5",
        },
      }
    ],
  },
  {
    "path": "samples/JGroups/src/ImmutableReference.java",
    "date": "2015-12-10",
    "md5": "48ca3c72fb9a65c771a321222f118b88",
    "type": "file",
    "mime_type": "text/plain",
    "size": "1838",
    "programming_language": "Java",
    "sha1": "30f56b876d5576d9869e2c5c509b08db57110592",
    "name": "ImmutableReference.java",
    "extension": ".java",
    "file_type": "ASCII text",
    "is_text": true,
    "license_expression": "lgpl-2.1-plus and lgpl-2.0-plus",
    "is_source": true,
    "copyrights": [{
      "end_line": 5,
      "start_line": 2,
      "holder": "Red Hat, Inc.",
      "statement": "Copyright 2010, Red Hat, Inc."
    }],
    "licenses": [
      { "detected": true,
        "key": "lgpl-2.1-plus",
        "category": "Copyleft Limited",
        "homepage_url": "http://www.gnu.org/licenses/old-licenses/lgpl-2.1-standalone.
→html",
        "start_line": 7,
        "end_line": 10,
        "short_name": "LGPL 2.1 or later",
        "owner": "Free Software Foundation (FSF)",
        "dejacode_url": "https://enterprise.dejacode.com/license_library/Demo/lgpl-2.1-
→plus/",
        "detection_score": 100.0,
        "external_reference": {
          "url": "http://spdx.org/licenses/LGPL-2.1+",
          "source": "spdx.org",
          "key": "LGPL-2.1+"
        }
      },
      { "concluded": true,
        "key": "lgpl-2.0-plus",
        "short_name": "LGPL 2.0 or later",
        "homepage_url": "http://www.gnu.org/licenses/old-licenses/lgpl-2.0.html",
        "end_line": 20,
        "dejacode_url": "https://enterprise.dejacode.com/license_library/Demo/lgpl-2.0-
→plus/",
```

```
          "text_url": "http://www.gnu.org/licenses/old-licenses/lgpl-2.0-standalone.html
→",
          "owner": "Free Software Foundation (FSF)",
          "start_line": 12,
          "detection_score": 47.46,
          "category": "Copyleft Limited",
          "external_reference": {
            "url": "http://spdx.org/licenses/LGPL-2.0+",
            "source": "spdx.org",
            "key": "LGPL-2.0+"
          }
        }
      ],
    },
    {
      "path": "samples/arch/zlib.tar.gz",
      "file_type": "gzip compressed data, last modified: Wed Jul 15 11:08:19 2015, from
→Unix",
      "date": "2015-12-10",
      "is_binary": true,
      "md5": "20b2370751abfc08bb3556c1d8114b5a",
      "sha1": "576f0ccfe534d7f5ff5d6400078d3c6586de3abd",
      "name": "zlib.tar.gz",
      "extension": ".gz",
      "size": 28103,
      "type": "file",
      "is_archive": true,
      "mime_type": "application/x-gzip",
      "packages": [
        {
          "type": "plain tarball"
        }
      ],
    }
  ]
}
```

AboutCode Manager

As a primary GUI for data review and integration, AboutCode Manager will need to be fluent in ABC Data to read/write ABC Data locally and remotely through API from several sources.

The short term changes would include:

- Support reading ABC Data from ScanCode

- Writing ABC Data, adding conclusions as related objects in the proper lists

New and Future tools

- TraceCode: would likely specify low level attributes for files (such as debug symbols, etc) and how files are related from devel to deploy and back.

- VulnerableCode: would likely specify a new Vulnerability object and the related attributes and may track several identifiers to the NIST NVD CPE and CVE.

- DeltaCode: would likely specify attributes to describe the changes between codebases, files, packages.

Copyright (c) 2016 nexB Inc.

3.2 Archived Pages

Archived aboutcode pages.

3.2.1 Table of Contents

Contributor Project Ideas (old)

Welcome to AboutCode!

AboutCode is a project to uncover data … about software code:

- where does the code come from? which software package?

- what's is its license? copyright?

- is the code secure, maintained, well coded?

All these are questions that are important to find answers to: there are million of free and open source software components available on the web.

Knowing where a software package comes from, if it is vulnerable and what's its licensing should be a problem of the past such that everyone can safely consume more free and open source software.

Join us to make it so!

Our tools are used to help detect and report the origin and license of source code, packages and binaries as well as discover software and package dependencies, and track security vulnerabilities, bugs and other important software package attributes. This is a suite of command line tools, web-based and API servers and desktop applications.

Table of Contents

- *AboutCode data server*

- *VulnerableCode Package security vulnerability data feed (and scanner)*

- *Integrate the license expression library in ScanCode (Python) and AboutCode Manager (JScript)*

- *High volume matching automatons and data structures*

- *ScanCode scan deduction*

- *License and copyright detection benchmark*

- *Improved copyright parsing and speed in ScanCode*

- *Transparent archive extraction in ScanCode*

- *Port ScanCode to Python 3*

- *Automated Docker, containers and VM images static package analysis*

- *Static analysis of binaries for build tracing in TraceCode*

- *Create Linux distro packages for ScanCode*

- *Package URL implementations in many programming languages*

- *DependentCode: a mostly universal Package dependencies resolver*

AboutCode projects are...

- **ScanCode Toolkit** a popular command line tool to scan code for licenses, copyrights and packages, used by many organizations and FOSS projects, small and large.

- **AboutCode Manager** a JavaScript, Electron-based desktop application to review scan results and document your conclusions

- **AboutCode Toolkit** a set of command line tools to document and inventory known packages and licenses and generate attribution docs

- TraceCode Toolkit: a set of command line tools to find which source code is used to create a compiled binary

- DeltaCode Toolkit: a new command line tool to compare codebases based on scan and determine if and where there are material differences that affect licensing

- VulnerableCode Server: a new server-side application to track package vulnerabilities

- AboutCode Server: a new server-side application to run and organize scans and ABC data (formerly ScanCode server)

- ConAn: a command line tool to analyze the code in Docker and container images

- license-expression: a library to parse and render boolean license expression (such as SPDX)

- Other new tools for source and binary code matching/search and package inventories.

We also work closely with other orgs and projects:

- purl aka. Package URLs https://github.com/package-url which is an emerging standard to reference software packages of all types.

- SPDX.org aka. Software Package Data Exchange, a spec to document the origin and licensing of packages

Contact

Join the chat online or by IRC at https://gitter.im/aboutcode-org/discuss introduce yourself and start the discussion!

For personal issues, you can contact the primary org admin directly: @pombredanne and pombredanne@gmail.com

Please ask questions the smart way: http://www.catb.org/~esr/faqs/smart-questions.html

Technology

Discovering the origin of code is a vast topic. We primarily use Python for this and some C/C++ (and eventually Rust) for performance sensitive code and Electron/JavaScript for GUI. We are open to using any other language within reason.

Our domain includes text analysis and processing (for instance for copyrights and licenses), parsing (for package manifest formats), binary analysis (to detect the origin and license of binaries, which source code they come from, etc) as well as web based tools and APIs (to expose the tools and libraries as web services) and low-level data structures for efficient matching (such as Aho- Corasick and other automata)

About your project application

We expect your application to be in the range of 1000 words. Anything less than that will probably not contain enough information for us to determine whether you are the right person for the job. Your proposal should contain at least the following information, plus anything you think is relevant:

- Your name

- Title of your proposal

- Abstract of your proposal

- Detailed description of your idea including explanation on why is it innovative and what it will contribute

- hint: explain your data structures and the main processing flows in details.

- Description of previous work, existing solutions (links to prototypes, bibliography are more than welcome)

- Mention the details of your academic studies, any previous work, internships

- Relevant skills that will help you to achieve the goal (programming languages, frameworks)?

- Any previous open-source projects (or even previous GSoC) you have contributed to and links.

- Do you plan to have any other commitments during GSoC that may affect your work? Any vacations/holidays? Will you be available full time to work on your project? (Hint: do not bother applying if this is not a serious full time commitment)

Join the chat online or by IRC at https://gitter.im/aboutcode-org/discuss introduce yourself and start the discussion!

You need to understand something about open source licensing or package managers or code and binaries static analysis or low level data structures. The best way to demonstrate your capability would be to submit a small patch ahead of the project selection for an existing issue or a new issue.

We will **always** consider and prefer a project submissions where you have submitted a patch over any otherr submission without a patch.

Our Project ideas

Here is a list of candidate project ideas for your consideration. Your own ideas are welcomed too! Please chat about them to increase your chances of success!

Note that there is NO specific order in this list!

AboutCode data server

This project is to futher and evolve the ScanCode server (was started last year as a 2017 GSoC project) and rename it as the AboutCode server.

The features of this updated server would be:

- Store any ABC data including ScanCode scans See *AboutCode Data Structure (ABCD)*

- Organize the data in projects (including possibly user-private projects)

- Execute selectively AboutCode tools such as ScanCode-toolkit, AboutCode-toolkit, etc.

- Integrate the storage and retrieval of scans and ABC data with the AboutCode Manager app through the JSON API.

- Add a Github integration to scan/run an ABC tool on commit with webhooks.

    - Bonus feature is to scan based on a received tweet of similar IRC or IM integration.

- **Tech**

    - Python 2, Django, PostgreSQL, DRF, JavaScript, Electron

- **URLS**

    - https://github.com/nexB/scancode-server

    - https://github.com/nexB/aboutcode-manager

    - https://github.com/nexB/aboutcode-toolkit

    - https://github.com/nexB/scancode-toolkit

- **Mentors**

    - @steven-esser https://github.com/steven-esser

    - @tdruez https://github.com/tdruez

VulnerableCode Package security vulnerability data feed (and scanner)

This project is to futher and evolve the VulnerableCode server and and software package vulnerabilities data aggregator.

VulnerableCode was started last year as a 2017 GSoC project. Its goal is to collect and aggregate vulnerabilities data and provide semi-automatic correlation. In the end it should provide the basis to report vulnerabilities alerts found in packages identified by ScanCode.

This is not trivial as there are several gaps in the CVE data and how they relate to packages as they are detected by ScanCode or else.

The features and TODO for this updated server would be:

- Aggregate more and new packages vulnerabilities feeds,

- Automating correlation: add smart relationship detection to infer new relatiosnhips between available packages and vulnerabilities from mining the graph of existing relations.

- Create a ScanCode plugin to report vulnerabilities with detected packages using this data.

- Integrate API lookup on the server withe the AboutCode Manager UI

- Create a UI and model for community curation of vulnerability to package mappings, correlations and enhancements.

- **Tech**

    - Python 2, Django, PostgreSQL, DRF, JavaScript, Electron

- **URLS**

    - https://github.com/nexB/vulnerablecode

    - https://github.com/nexB/aboutcode-manager

    - https://github.com/nexB/scancode-toolkit

    - Other interesting pointers:

        * https://github.com/cve-search/cve-search

        * https://github.com/jeremylong/DependencyCheck/

        * https://github.com/victims/victims-cve-db

        * https://github.com/rubysec/ruby-advisory-db

        * https://github.com/future-architect/vuls

        * https://github.com/coreos/clair

        * https://github.com/anchore/anchore/

        * https://github.com/pyupio/safety-db

        * https://github.com/RetireJS/retire.js

        * and many more including Linux distro feeds

- **Mentors**

    - @steven-esser https://github.com/steven-esser

    - @JonoYang https://github.com/JonoYang

    - @pombredanne https://github.com/pombredanne

Integrate the license expression library in ScanCode (Python) and AboutCode Manager (JScript)

In GSoC 2017, this Python library was ported to JavaScript using Transcrypt.

The goal of this project is to add support for license expressions in multiple projects and evolve the license expression library as needed:

- in Python:

    - the SPDX Python library

    - the ScanCode toolkit. This also include the proper detection of license expressions in SPDX-License-Identifier tags.

    - the AboutCode toolkit

- in JavaScript:

    - the AboutCode Manager

- in both languages in the core license expression proper, add support for a built-in mode for strict SPDX expressions

- **Tech**

    - Python, JavaScript

- **URLS**

    - https://github.com/nexB/license-expression

    - https://github.com/bastikr/boolean.py

    - https://github.com/nexB/aboutcode-manager

    - https://github.com/nexB/aboutcode-toolkit

    - https://github.com/nexB/scancode-toolkit

    - https://github.com/spdx/tools-python

- **Mentors**

    - @JonoYang https://github.com/JonoYang

    - @steven-esser https://github.com/steven-esser

High volume matching automatons and data structures

MatchCode will provide ways to efficiently match actual code against a large stored indexes of open source code.

To enable this, we need to research and create efficient and compact data structures that are specialized for the type of data we lookup. Given the volume to consider (typically multi billion values indexed) there are special considerations to have compact and memory efficient dedicated structures (rather than using a general purpose DB or Key/value pair store) that includes looking at automata, and memory mapping. This types of data structures should be implemented in Rust as a preference (though C/C++ is OK) and include Python bindings.

There are several areas to research and implement:

- A data structure to match efficiently a batch of fix-width checksums (e.g. SHA1) against a large index of such checksums, where each checksum points to one or more files or packages. A possible direction is to use finite state transducers or specialized B-tree indexes. Since when a codebase is being matched there can be millions of lokkups to do, the batch matching is preferred.

- A data structure to match efficiently a batch of fix-width byte strings (e.g. LSH) against a large index of such LSH within a fixed hamming distance, where each points to one or more files or packages. A possible direction is to use finite state transducers (possibly weighted), specialized B-tree indexes or multiple hash-like on-disk tables.

- A memory-mapped Aho-Corasick automaton to build large batch tree matchers. Available Aho-Corasick automaton may not have a Python binding or may not allow memory-mapping (like pyahocorasick we use in ScanCode). The volume of files we want to handle requires to reuse, extend or create aspecialized tree/paths matching automatons that can handle eventually billions of nodes and are larger than the available RAM. A possible direction is to use finite state transducers (possibly weighted).

- Feature hashing research: we deal with manyt "features" and hashing to limit the number and size of the each features seems to be a valuable thing. The goal is to research feature hashing with short hashes (15, 16 and 32 bits) and evaluate if this leads to acceptable fasle-positive and loss of accuracy in the context of the data structures mentioned above.

Then using these data structures, the project should create a system for matching code as a Python-based server exposing a simple API. This is a green field project.

- **Tech**

  - Rust, Python

- **URLS**

  - https://github.com/nexB/scancode-toolkit-contrib for samecode fingerprints drafts.

  - https://github.com/nexB/scancode-toolkit for commoncode hashes

- **Mentors**

  - @pombredanne https://github.com/pombredanne

ScanCode scan deduction

The goal of this project is to take existing scan and match results and infer summaries and deduction at a higher level, such as the licensing or origin of a whole directory tree. This should be implemented as a set of ScanCode plugins

- **Tech**

  - Python

- **URLS**

  - https://github.com/nexB/scancode-toolkit/issues/426

  - https://github.com/nexB/scancode-toolkit/issues/377

- **Mentors**

  - @pombredanne https://github.com/pombredanne

  - @JonoYang https://github.com/JonoYang

License and copyright detection benchmark

Compare ScanCode runtimes with Fossology, licensee, LicenseFinder, license- check, ninka, slic, LiD and others. This project is to create a comprehensive test suite and a benchmark for several FOSS open source license and copyright detection engines, establish mappings between the different conventions they use for license identification and evaluate and publish the results of detection accuracy and precision.

Note that this not only about the speed of scanning: the performance and time taken is accessory and a nice to have as a result. What matters is benchmarking the accuracy of the license detection:

1. is the right license detected and how correct is this detection?

2. when a license is detected is the correct exact text matched and returned?

So what is needed is a (large) test set of files.

Then establishing a ground truth for reference e.g. detecting then reviewing manually possibly with ScanCode to set up the baseline that will be used to compare all the scanners.

Then run the other tools and ScaCode to see how well they perform and of course establish a mapping of license identifiers: each tool may use different license ids so we need to map these to the ids used in the test baseline (e.g. the scancode license keys): all this has to be built, possibly reusing some or all of the scancode tests and lacing in all the tests from the other tools and adding more as needed.

- **Tech**

- – Python

- **Mentors**

  - – @mjherzog https://github.com/mjherzog

  - – @pombredanne https://github.com/pombredanne

Improved copyright parsing and speed in ScanCode

Copyright detection is the slowest scanner in ScanCode. It is based on NLTK part of speech tagging and a copyright grammar.

The goal of this project is to refactor Copyright detection for speed and simplicity possibly implementaing a new parser (PEG?, etc) or reimplementing core elements in Rust with a Python binding.

This would include also keeping track of line numbers and offsets where copyrights are found.

This would likely require either replacing or enhancing NLTK which is used as a natural language parser.

- **Tech**

  - – Python, Rust

- **URLS**

  - – https://github.com/nexB/scancode-toolkit/tree/develop/src/cluecode

- **Mentor**

  - – @JonoYang https://github.com/JonoYang

Transparent archive extraction in ScanCode

ScanCode archive extraction is currently done with a separate command line invocation. The goal of this project is to integrate archive extraction transparently into the ScanCode scan loop. This would be using the new plugins architecture.

- **Tech**

  - – Python

- **URLS**

  - – https://github.com/nexB/scancode-toolkit/issues/14

- **Mentor**

  - – @pombredanne https://github.com/pombredanne

Port ScanCode to Python 3

ScanCode runs only on Python 2.7 today. The goal of this project is to port ScanCode to support both Python 2 and Python 3.

- **Tech**

  - – Python

- **URLS**

  - – https://github.com/nexB/scancode-toolkit/issues/295

- **Mentor**

    - @pombredanne https://github.com/pombredanne

Automated Docker, containers and VM images static package analysis

The goal of this project is to further the Conan container static analysis tool to effectively support proper inventory of installed packages without running the containers.

This includes determining which packages are installed in Docker layers for RPMs, Debian or Alpine Linux. And this woudl eventually require the integration of ScanCode.

- **Tech**

    - Python, Go?

- **URLS**

    - https://github.com/pombredanne/conan

    - https://github.com/nexB/scancode-toolkit

- **Mentor**

    - @pombredanne https://github.com/pombredanne

Static analysis of binaries for build tracing in TraceCode

TraceCode does system call tracing only today.

- The primary goal of this project is to do the same using symbol, debug symbol or string matching to accomplish something similar using static analysis.

- This project also would cover updating TraceCode to use the Click comamnd line toolkit (like for ScanCode).

- Finally thsi project should improve the tracing of the lifecycle of file descriptors in TraceCode build. We need to improve how TraceCode does system call tracing by improving the way we track open/close file descriptors in the trace to reconstruct the lifecycle of a traced file.

- **Tech**

    - Python, Linux

- **URLS**

    - https://github.com/nexB/tracecode-toolkit for the existing non-static tool

    - https://github.com/nexB/scancode-toolkit-contrib for the work in progress on binaries/symbols parsers/extractors

- **Mentor**

    - @pombredanne https://github.com/pombredanne

Create Linux distro packages for ScanCode

The goal of this project is to ensure that we have proper packages for Linux distros for ScanCode.

The first step is to debundle pre-built binaries that exist in ScanCode such that they come either from system-packages or pre-built Python wheels. This covers libarchive, libmagic and a few other native libraries.

The next step is to ensure that all the dependencies from ScanCode are also available as distro packages.

The last step is to create proper distro packages for RPM, Debian, Nix and GUIX, Alpine, Arch and Gentoo and also an AppImage.org package as well as a proper Docker image and eventually submit these package to the distros.

As a bonus, the same could then be done for AboutCode toolkit and TraceCode.

This requires a good understanding of packaging and Python.

- **Tech**

    - Python, Linux

- **URLS**

    - https://github.com/nexB/scancode-toolkit/issues/487

    - https://github.com/nexB/scancode-toolkit/issues/469

- **Mentor**

    - @pombredanne https://github.com/pombredanne

Package URL implementations in many programming languages

We have a purl implmentation in Python, Go and possibly Java today.

The goal of this project is to create multiple parsers and builders in several programming languages:

- JavaScript, Rust, R, Perl, Ruby, C/C++, Racket, etc.

- **Tech**

    - Many!

- **URLS**

    - https://github.com/package-url

    - https://fosdem.org/2018/schedule/event/purl/

- **Mentors**

    - @pombredanne https://github.com/pombredanne

DependentCode: a mostly universal Package dependencies resolver

The goal of this project is to create a tool for mostly universal package dependencies resolution using a SAT solver that should leverage the detected packages from ScanCode and the Package URLs and could provide a good enough way to resolve package dependencies for many system and application package formats. This is a green field project.

- **Tech**

    - Python, C/C++, Rust, SAT

- **URLS**

- – https://github.com/package-url

  – https://fosdem.org/2018/schedule/event/purl/

- **Mentors**

  – @pombredanne https://github.com/pombredanne

3.3 License

AboutCode includes documents that are dedicated to the Public Domain using the Creative Commons CC0 1.0 Universal (CC0 1.0) Public Domain Dedication: http://creativecommons.org/publicdomain/zero/1.0/

G

Google Season of Docs 2020, 61
Google Season of Docs 2021, 57