sphinx-test-1.0

Release 1.0

Nov 04, 2019

Contents

CHAPTER 1

Aboutcode Projects

1.1 Scancode-Workbench Documentation

ScanCode Workbench allows you take the scan results from the ScanCode Toolkit and create a software inventory annotated with your summaries or conclusions (we call these Conclusions) at any levels of the codebase you choose.

The attributes you add (e.g., Name, Version, Owner, License Expression, Copyright) to your Conclusion about a single package or file – or a higher-level group of packages and/or files – can then be exported to a JSON or SQLite file. In addition, Conclusions created in ScanCode Workbench can be exported to DejaCode.

1.1.1 Basics

Scancode Workbench Views

Directory Tree

An interactive directory tree is always present on the left side of the application. The tree is expandable and collapsible. This allows the user to navigate the codebase structure. If a directory is selected, only that directory and its sub-files and folders will be shown in the view. Similarly, if a single file is selected, only information for that selected file will be shown.

Table View

In the table view, the available clues detected by ScanCode are shown in a tabular format. A user can see provenance clues such as license and copyright information detected by ScanCode. A user can also see the file information (e.g. file type, file size, etc) and package information (package type, primary language of package) that was detected. The columns can be sorted as well as shown or hidden based on what the user's preferences. Searching for specific clues (license names, copyrights, etc.) is also available in this view.

Chart Summary View

With the chart summary view, a user can select a node in the directory tree (i.e., a directory, folder or file) and display a horizontal bar chart listing the values identified in the scanned codebase – that is, the clues detected by ScanCode Toolkit – for a number of different attributes. The attributes are a subset of the columns displayed in the table view, and can be selected by clicking the dropdown at the top of the view. The chart displays the full range of values for the selected directory tree node and attribute and the number of times each value occurs in the scanned codebase.

Building Requirements

Linux

- Python 2.7
- Node.js version 6.x or later
- npm 3.10.x or later but <= 5.2.0 (run `npm install npm@5.2.0 -g`)

MacOS

- Python 2.7
- Node.js >=6.x or later but <=8.9.4
- npm 3.10.x or later but <= 5.2.0 (run `npm install npm@5.2.0 -g`)
- Command Line Tools for Xcode (run `xcode-select --install to install`)

Windows

- Node.js 6.x or later
- npm 3.10.x or later but <= 5.2.0 `(run npm install npm@5.2.0 -g)`
- Python v2.7.x
  - Make sure your Python path is set. To verify, open a command prompt and type `python --version`. Then, the version of python will be displayed.
- Visual C++ Build Environment:
  - Either:
  - Option 1: Install Visual C++ Build Tools 2015 (or modify an existing installation) and select Common Tools for Visual C++ during setup. This also works with the free Community and Express for Desktop editions.
  - Option 2: Visual Studio 2015 (Community Edition or better)
  - Note: Windows 7 requires .NET Framework 4.5.1
  - Launch cmd, `npm config set msvs_version 2015`

ScanCode Workbench Platform Support

Our approach for platform support is to focus on one primary release for each of Linux, MacOS and Windows. The Priority definitions are:

1. Primary - These are the primary platforms for build/test/release on an ongoing basis.
2. Secondary - These are platforms where the primary ScanCode Workbench release for the corresponding OS Group should be forward-compatible, e.g., Windows 7 build should work on Windows 10. Issues reported and traced to a Secondary platform may not be fixed.
3. Tertiary - These are any other platforms not listed as Primary or Secondary. In these cases, we will help users help themselves, but we are likely not to fix Issues that only surface on a Tertiary platform.

OS Group	Desktop OS Version	Arch	Priority	Notes
Windows	Windows 7 SP1	x64	1	
Windows	Windows 10 SP?	x64	2	
MacOS	10.9 Mavericks	x64	1	
MacOS	10.10 Yosemite	x64	2	
MacOS	10.11 El Capitan	x64	2	
MacOS	10.12 Sierra	x64	2	
Linux Deb	Ubuntu 12.04	x64	1	From Electron Docs: The prebuilt ia32 (i686) and x64 (amd64) binaries of Electron are built on Ubuntu 12.04.
Linux Deb	Ubuntu 14.xx	x64	2	Verified to be able to run the prebuilt binaries of Electron.
Linux Deb	Ubuntu 16.xx	x64	2	Verified to be able to run the prebuilt binaries of Electron.
Linux	Fedora 21	x64	2	Verified to be able to run the prebuilt binaries of Electron.
Linux	Debian 8	x64	2	Verified to be able to run the prebuilt binaries of Electron.
Linux RH	CentOS 7.xx	x64	?	
Linux RH	RHEL 7.xx	x64	?	

Electron Supported Platforms

https://electronjs.org/docs/tutorial/support#supported-platforms

The following platforms are supported by Electron:

MacOS

Only 64-bit binaries are provided for MacOS, and the minimum MacOS version supported is MacOS 10.9.

Windows

Windows 7 and later are supported, while older operating systems are not supported (and do not work). Both ia32 (x86) and x64 (amd64) binaries are provided for Windows. Please note: the ARM version of Windows is not supported for now.

Linux

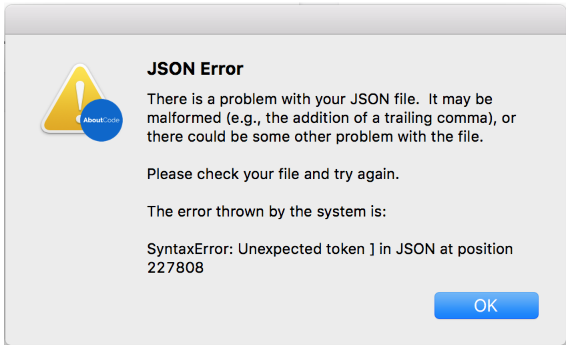
The prebuilt ia32 (i686) and x64 (amd64) binaries of Electron are built on Ubuntu 12.04, and the ARM binary is built against ARM v7 with hard-float ABI and NEON for Debian Wheezy.

Whether the prebuilt binary can run on a distribution depends on whether the distribution includes the libraries that Electron is linked to on the building platform, so only Ubuntu 12.04 is guaranteed to work, but the following platforms are also verified to be able to run the prebuilt binaries of Electron:

- Ubuntu 12.04 and later
- Fedora 21
- Debian 8

Check for Errors in the Developer Tools

When an unexpected error occurs in ScanCode Workbench, you will normally see a dialog message which provides details about the error and allows you to create an issue.



JSON Error

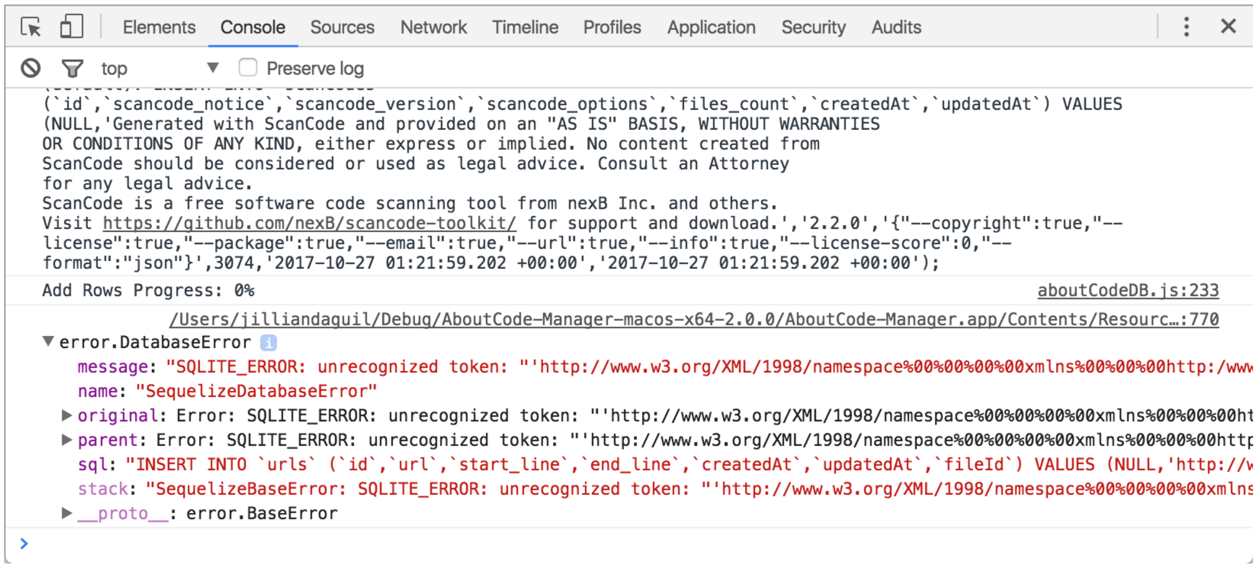
There is a problem with your JSON file. It may be malformed (e.g., the addition of a trailing comma), or there could be some other problem with the file.

Please check your file and try again.

The error thrown by the system is:

SyntaxError: Unexpected token] in JSON at position 227808

If you can reproduce the error, use this approach to get the stack trace and report the issue. Open the Developer Tools with `Ctrl+Shift+I` or `Alt+Cmd+I`. From there, click the Console tab. Include the error that is logged in the issue in a code block or a file attachment.



```
(`id`,`scancode_notice`,`scancode_version`,`scancode_options`,`files_count`,`createdAt`,`updatedAt`) VALUES
(NULL,'Generated with ScanCode and provided on an "AS IS" BASIS, WITHOUT WARRANTIES
OR CONDITIONS OF ANY KIND, either express or implied. No content created from
ScanCode should be considered or used as legal advice. Consult an Attorney
for any legal advice.
ScanCode is a free software code scanning tool from nexB Inc. and others.
Visit https://github.com/nexB/scancode-toolkit/ for support and download.','2.2.0','{"--copyright":true,"--
license":true,"--package":true,"--email":true,"--url":true,"--info":true,"--license-score":0,"--
format":"json"}',3074,'2017-10-27 01:21:59.202 +00:00','2017-10-27 01:21:59.202 +00:00');
Add Rows Progress: 0%                                                        aboutCodeDB.js:233
          /Users/jilliandaguil/Debug/AboutCode-Manager-macos-x64-2.0.0/AboutCode-Manager.app/Contents/Resourc…:770
▼error.DatabaseError
    message: "SQLITE_ERROR: unrecognized token: "'http://www.w3.org/XML/1998/namespace%00%00%00%00xmlns%00%00%00http:/www
    name: "SequelizeDatabaseError"
  ▶original: Error: SQLITE_ERROR: unrecognized token: "'http://www.w3.org/XML/1998/namespace%00%00%00%00xmlns%00%00%00ht
  ▶parent: Error: SQLITE_ERROR: unrecognized token: "'http://www.w3.org/XML/1998/namespace%00%00%00%00xmlns%00%00%00http
    sql: "INSERT INTO `urls` (`id`,`url`,`start_line`,`end_line`,`createdAt`,`updatedAt`,`fileId`) VALUES (NULL,'http://w
    stack: "SequelizeBaseError: SQLITE_ERROR: unrecognized token: "'http://www.w3.org/XML/1998/namespace%00%00%00%00xmlns
  ▶__proto__: error.BaseError
>
```

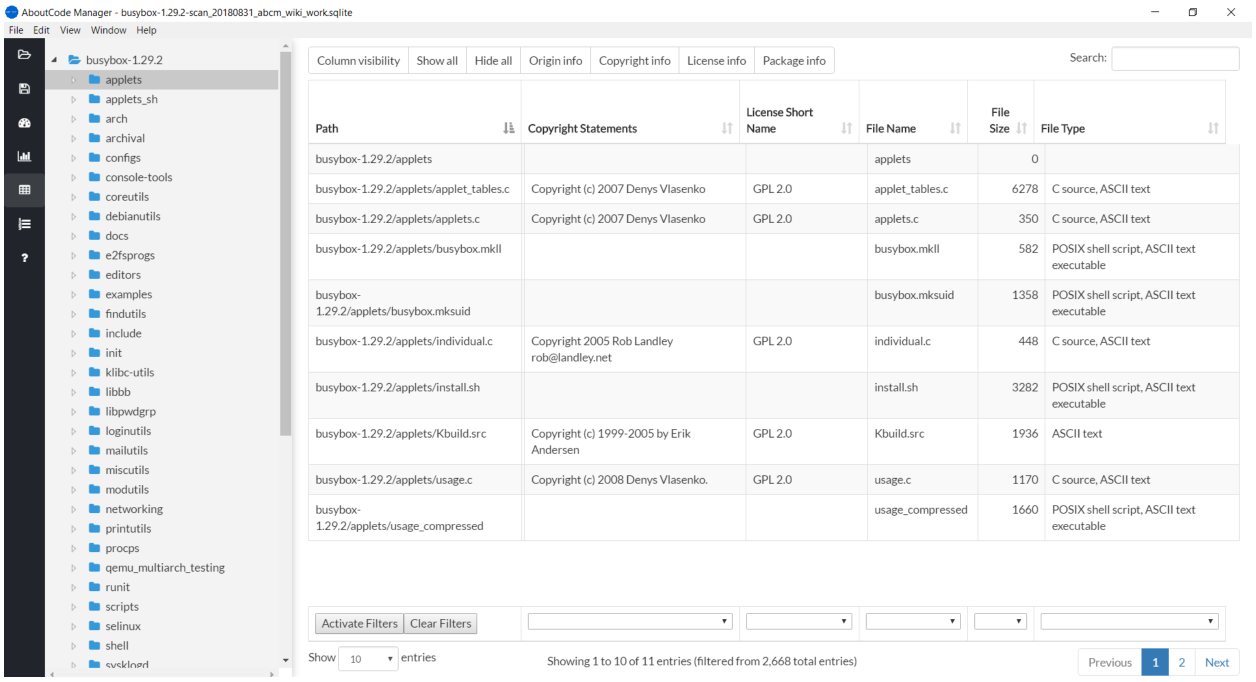
1.1.2 Tutorials

Creating Conclusions

A Conclusion in ScanCode Workbench refers to the documentation of your analysis and conclusions about the name, version, owner, copyright, license expression and other attributes for a single software package or file or – if you conclude these attributes are shared by a group of packages and/or files – for that group of packages/files.

You can record your Conclusions throughout the codebase you're analyzing, at any level of the codebase (i.e., nodes in the directory tree representing the codebase) you think will best reflect the results of your analysis.

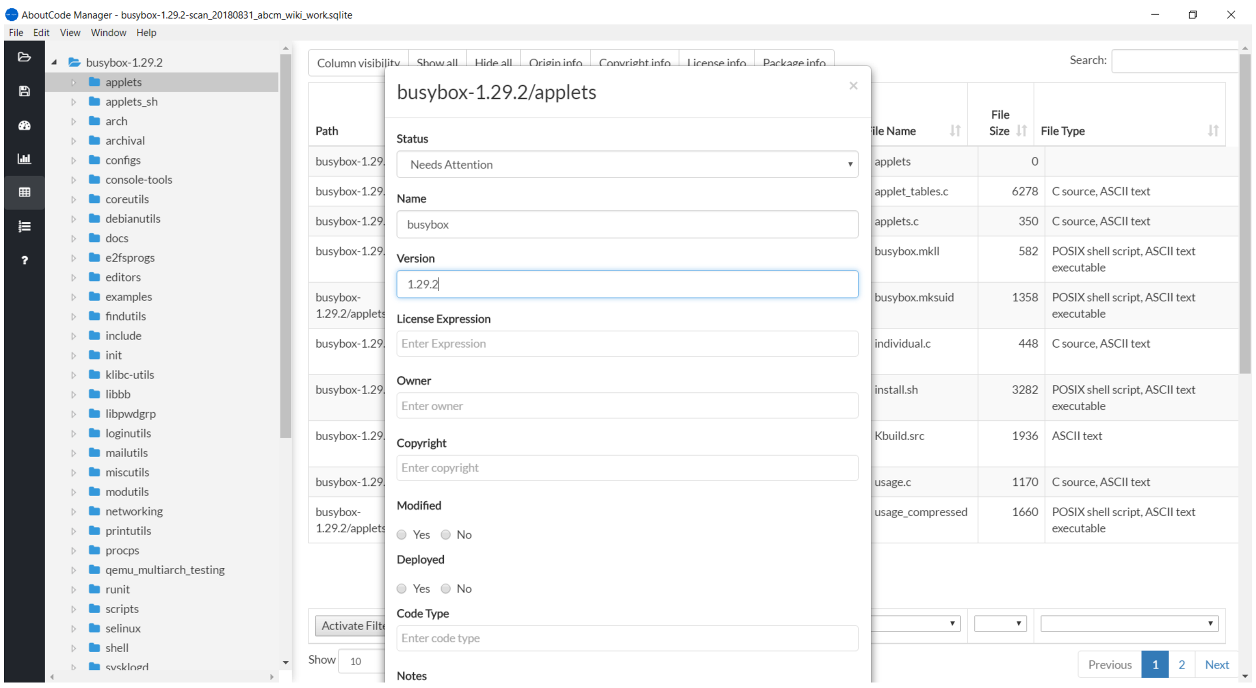
To create a Conclusion, begin by navigating to the `ScanDataTable` view.



Path	Copyright Statements	License Short Name	File Name	File Size	File Type
busybox-1.29.2/applets			applets	0	
busybox-1.29.2/applets/applet_tables.c	Copyright (c) 2007 Denys Vlasenko	GPL 2.0	applet_tables.c	6278	C source, ASCII text
busybox-1.29.2/applets/applets.c	Copyright (c) 2007 Denys Vlasenko	GPL 2.0	applets.c	350	C source, ASCII text
busybox-1.29.2/applets/busybox.mkll			busybox.mkll	582	POSIX shell script, ASCII text executable
busybox-1.29.2/applets/busybox.mksuid			busybox.mksuid	1358	POSIX shell script, ASCII text executable
busybox-1.29.2/applets/individual.c	Copyright 2005 Rob Landley rob@landley.net	GPL 2.0	individual.c	448	C source, ASCII text
busybox-1.29.2/applets/install.sh			install.sh	3282	POSIX shell script, ASCII text executable
busybox-1.29.2/applets/Kbuild.src	Copyright (c) 1999-2005 by Erik Andersen	GPL 2.0	Kbuild.src	1936	ASCII text
busybox-1.29.2/applets/usage.c	Copyright (c) 2008 Denys Vlasenko.	GPL 2.0	usage.c	1170	C source, ASCII text
busybox-1.29.2/applets/usage_compressed			usage_compressed	1660	POSIX shell script, ASCII text executable

Showing 1 to 10 of 11 entries (filtered from 2,668 total entries)

In the directory tree on the left, choose the directory, package or file you want to annotate, right-click that node, and select `Edit Conclusion` in the menu that appears. This will display a form with the path to that node displayed at the top of the form, and a series of attribute names (e.g., `Status`, `Name`, `Version`, `License Expression`, `Owner`, `Copyright`) and associated textboxes, dropdowns or radio buttons to record your input.



busybox-1.29.2/applets

Status: Needs Attention

Name: busybox

Version: 1.29.2

License Expression: Enter Expression

Owner: Enter owner

Copyright: Enter copyright

Modified: Yes No

Deployed: Yes No

Code Type: Enter code type

Notes

Path	File Name	File Size	File Type
busybox-1.29.	applets	0	
busybox-1.29.	applet_tables.c	6278	C source, ASCII text
busybox-1.29.	applets.c	350	C source, ASCII text
busybox-1.29.	busybox.mkll	582	POSIX shell script, ASCII text executable
busybox-1.29.2/applets	busybox.mksuid	1358	POSIX shell script, ASCII text executable
busybox-1.29.	individual.c	448	C source, ASCII text
busybox-1.29.	install.sh	3282	POSIX shell script, ASCII text executable
busybox-1.29.	Kbuild.src	1936	ASCII text
busybox-1.29.	usage.c	1170	C source, ASCII text
busybox-1.29.2/applets	usage_compressed	1660	POSIX shell script, ASCII text executable

Conclusion Definitions

The following fields are available when creating a Conclusion:

Conclusion Field	Description
Status	Used to document status of Conclusion creation. The dropdown choices: **Analyzed** - A Conclusion has been created, **Needs Attention** - A Conclusion is flagged for further review, **Original Code** - A Conclusion references code from your organization, **Not Reporting** - A Conclusion that will not be reported and can be ignored.
Name	The primary name for your Conclusion (usually a file, directory or library name). A Conclusion can represent any software-related object. Any Conclusion can contain one or more sub-Conclusions. The combined Conclusion Name and Version must be unique.
Version	The Conclusion version number.
License Expression	This is the overall license (an individual license or combination of several licenses) for the Conclusion. The Conclusion form will populate a dropdown with any License Expression data detected by ScanCode when the imported scan was run, using the ScanCode License Expression syntax (e.g., `gpl-2.0` represents the GPL 2.0 license). The user can also manually add one or more License Expressions by typing in the textbox and then hitting the `Enter` key. The License Expression is intended to capture the facts of a license (primarily the text, as provided by its owner), as well as an organization's interpretation and policy regarding that license.
Owner	An Owner identifies the original creator (copyright holder) of the code covered by the Conclusion. If this Conclusion code is in its original, unmodified state, the Conclusion owner is associated with the original author/publisher. If this Conclusion code has been copied and modified, the Conclusion owner should be the owner that has copied and modified it.
Copyright	The Copyright notice that applies to the Conclusion code under the License Expression.
Modified	A Yes/No choice indicating whether the Conclusion code has been modified.
Deployed	A Yes/No choice indicating whether the Conclusion code has been deployed.
Code Type	The default choices are **Source**, **Binary**, **Mixed** and **Document**.
Notes	Any notes or comments the user wants to record.
Feature	The name of a product feature or codebase module that applies to the code covered by the Conclusion.
Purpose	The type of code covered by the Conclusion, e.g., Core, Test, Build, Documentation.
Programming Language	The primary language of the Conclusion code.
Homepage URL	The homepage URL for the Conclusion code.
Download URL	The download URL for the original Conclusion code.
License URL	The URL for the primary license represented by the License Expression.
Notice URL	The URL for the license notice that applies to the code covered by the Conclusion.

Importing and Exporting a JSON File

Import a ScanCode JSON File

- We have provided a set of sample scans that you can quickly review in ScanCode Workbench in order to get a sense of its functionality and the types of information captured by a scan. The samples are located at https://github.com/nexB/scancode-workbench/tree/develop/samples.
- To import a ScanCode JSON file:
  - Open the `File` menu and select `Import JSON File` (keyboard shortcut: `Ctrl+I` or `+I`).
  - In the dialog window that opens, navigate to the JSON file you want to import, select the file and click `Open`.
  - You will then be prompted to choose a filename and location to save the JSON file as a SQLite database file. Add a filename, select the folder in which you want to save the SQLite database file, and click `Save`.
  - ScanCode Workbench will then create a SQLite database file from your JSON file, indicated by the status message "Creating Database . . . "
  - Once the process has finished, the status message will be replaced by an expandable code tree and, to the right of the tree, a table displaying provenance information generated by ScanCode.

Export a JSON file

- To export a JSON file:
  - Select the `File` menu and then select `Export JSON File` (keyboard shortcut: `Ctrl+E` or `+E`).
  - In the dialog window that opens, add a name for the file, navigate to the directory in which you want to save the file and click `Save`.

License Policy support in ScanCode Workbench

ScanCode Workbench now has basic support for tracking and viewing license policies that have been applied to a `scancode-toolkit` scan. In order for things to work, your initial `scancode` scan must be run with the `--license-policy` option. You can read more about that here: license_policy_plugin.

The basics

While the license_policy_plugin can be customized with any number of custom fields and values, ScanCode Workbench currently only supports a pre-defined set of policy labels.

license_key	label
scancode_license_key	Approved License
scancode_license_key	Prohibited License
scancode_license_key	Recommended License
scancode_license_key	Restricted License

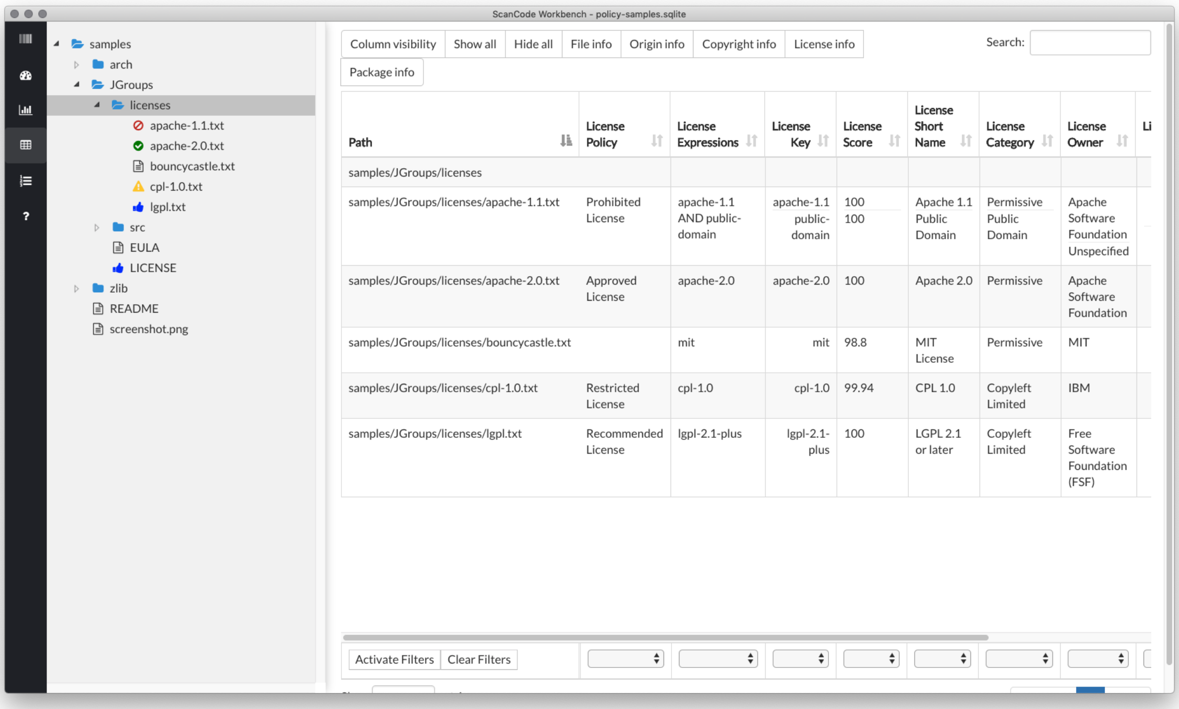
This means in order to take advantage of ScanCode Workbench's policy features, your `license-policy.yml` needs to have `license_key` and `label` fields at the very least.

Additionally, in order to take advantage of policy visualizations, `label` values must be one of the 4 above values: Approved License, Prohibited License, Recommended License or Restricted License. Later versions of ScanCode Workbench will eventually evolve to support more dynamic policy values.

Here is a simple example of a valid `license-policy.yml` file:

```
license_policies:
-   license_key: apache-2.0
    label: Approved License
-   license_key: apache-1.1
    label: Prohibited License
-   license_key: lgpl-2.1-plus
    label: Recommended License
-   license_key: cpl-1.0
    label: Restricted License
```

After running a scan with that particular `license-policy.yml` file, viewing the scan in ScanCode Workbench will look like the following:



Path	License Policy	License Expressions	License Key	License Score	License Short Name	License Category	License Owner	Li
samples/JGroups/licenses								
samples/JGroups/licenses/apache-1.1.txt	Prohibited License	apache-1.1 AND public-domain	apache-1.1 public-domain	100 100	Apache 1.1 Public Domain	Permissive Public Domain	Apache Software Foundation Unspecified	
samples/JGroups/licenses/apache-2.0.txt	Approved License	apache-2.0	apache-2.0	100	Apache 2.0	Permissive	Apache Software Foundation	
samples/JGroups/licenses/bouncycastle.txt		mit	mit	98.8	MIT License	Permissive	MIT	
samples/JGroups/licenses/cpl-1.0.txt	Restricted License	cpl-1.0	cpl-1.0	99.94	CPL 1.0	Copyleft Limited	IBM	
samples/JGroups/licenses/lgpl.txt	Recommended License	lgpl-2.1-plus	lgpl-2.1-plus	100	LGPL 2.1 or later	Copyleft Limited	Free Software Foundation (FSF)	

As you can see, files which have detected licenses that fall under a particular policy will be shown in the JSTree view with specific icons. This way, you are able to quickly see what files fall under a specific policy.

Additionally, policy details can be found in the scan data view in their own column: License Policy. This column has been added to both the “Origin” column set and “License info” column set.

Navigating the Chart Summary View

Display the view

Once you have a SQLite file loaded into ScanCode Workbench, displaying the Chart Summary View is easy:

1. Select a file or directory in the Tree View on the left.
2. Click the chart icon in the sidebar or open the `View` menu and select `Chart Summary View` (keyboard shortcut: `Ctrl+Shift+D` or `+Shift+D`).

Select an attribute

Use the dropdown at the top of the view to select the attribute you want to examine (e.g., `Copyright Statements`, `License Key`). These attribute values are detected from ScanCode, and can also be viewed in the Table View.

When you select an attribute, the Chart Summary View will automatically refresh to display a horizontal bar chart showing – in descending order of frequency – each value identified in the scanned codebase for the selected attribute and the number of times it occurs in the codebase. You can also see the value for a particular entry in the bar chart in a tooltip that appears when you move your cursor over the text on the left or the bar on the right.

Filter Chart Summary

You can further filter the summary results by choosing a specific directory or file in the Tree View. The chart will then only show results for that selected directory or file.

Opening and Saving a SQlite File

Open a SQLite File

- To open a SQLite File:
  - Select the `File` menu and then select `Open SQLite File` (keyboard shortcut: `Ctrl+O` or `+O`).
  - In the dialog window that opens, navigate to the SQLite file you want to open, select the file and click `Open`.

Save as a New SQLite File

- To save as a new SQLite file:
  - Select the `File` menu and then select `Save As New SQLite File` (keyboard shortcut: `Ctrl+S` or `+S`).
  - In the dialog window that opens, add a name for the file, navigate to the directory in which you want to save the file and click `Save`.

1.2 Deltacode Documentation

Welcome to Deltacode Documentation.

1.2.1 Comprehensive Installation

DeltaCode requires Python 2.7.x and is tested on Linux, Mac, and Windows. Make sure Python 2.7 is installed first.

System Requirements

- Hardware : DeltaCode will run best with a modern X86 processor and at least 1GB of RAM and 250MB of disk.
- Supported operating systems : DeltaCode should run on these OSes:
  1. Linux: on most recent 64-bit Linux distributions (32-bit distros are only partially supported),
  2. Mac: on recent Mac OSX (10.6.8 and up),
  3. Windows: on Windows 7 and up (32- or 64-bit) using a 32-bit Python.

Prerequisites

DeltaCode needs a Python 2.7 interpreter.

- **On Linux**:

  Use your package manager to install `python2.7`. If Python 2.7 is not available from your package manager, you must compile it from sources.

  For instance, visit https://github.com/dejacode/about-code-tool/wiki/BuildingPython27OnCentos6 for instructions to compile Python from sources on Centos.

- **On Windows**:

  Use the Python 2.7 32-bit (e.g. the Windows x86 MSI installer) for X86 regardless of whether you run Windows on 32-bit or 64-bit. **DO NOT USE Python X86_64 installer** even if you run 64 bit Windows.

  Download Python from this url: https://www.python.org/ftp/python/2.7.14/python-2.7.14.msi

  Install Python on the c: drive and use all default installer options. See the Windows installation section for more installation details.

- **On Mac**:

  Download and install Python from this url:

  https://www.python.org/ftp/python/2.7.14/python-2.7.14-macosx10.6.pkg

Installation on Linux and Mac

Download and extract the latest ScanCode release from: https://github.com/nexB/deltacode/releases/latest

Open a terminal in the extracted directory and run:

```
./deltacode --help
```

This will configure DeltaCode and display the command line help.

Installation on Windows

Download the latest ScanCode release zip file from: https://github.com/nexB/deltacode/releases/latest

- In Windows Explorer, select the downloaded DeltaCode zip and right-click.
- In the pop-up menu select 'Extract All. . . '
- In the pop-up window 'Extract zip folders' use the default options to extract.

- Once the extraction is complete, a new Windows Explorer window will pop-up.
- In this Explorer window, select the new folder that was created and right-click.
- In the pop-up menu select 'Properties'
- In the pop-up window 'Properties', select the Location value. Copy this in clipboard.
- Press the start menu button.
- In the search box type

```
cmd
```

- Select 'cmd.exe' listed in the search results.
- A new 'cmd.exe' window pops-up.
- In this window (aka. a command prompt), type this (this is 'cd' followed by a space)

```
cd
```

- then right-click in this window and select Paste. This will paste the path where you extracted DeltaCode.
- Press Enter.
- This will change the current location of your command prompt to the root directory where DeltaCode is installed.
- Then type

```
deltacode --help
```

- Press enter. This will configure your DeltaCode installation.
- Several messages are displayed followed by the deltacode command help.
- The installation is complete.

Un-installation

- Delete the directory in which you extracted DeltaCode.
- Delete any temporary files created in your system temp directory under a deltacode directory.

1.2.2 Deltacode Output: Format, Fields and Structure

```
Usage: deltacode [OPTIONS]

  Identify the changes that need to be made to the 'old' scan file (-o or --old)
  in order to generate the 'new' scan file (-n or --new).  Write the results to
  a .json file (-j or --json-file) at a user-designated location.  If no file
  option is selected, print the JSON results to the console.

Options:
  -h, --help                Show this message and exit.
  --version                 Show the version and exit.
  -n, --new PATH            Identify the path to the "new" scan file [required]
  -o, --old PATH            Identify the path to the "old" scan file [required]
  -j, --json-file FILENAME  Identify the path to the .json output file
  -a, --all-delta-types     Include unmodified files as well as all changed
```

(continues on next page)

(continued from previous page)

```
                                files in the .json output.  If not selected, only
                                changed files are included.
```

Output Formats

DeltaCode provides two output formats for the results of a DeltaCode codebase comparison: `JSON` and `CSV`.

The default output format is `JSON`. If the command-line input does not include an output flag (`-j` or `--json-file`) and the path to the output file, the results of the DeltaCode comparison will be displayed in the console in `JSON` format. Alternatively, the results will be saved to a `.json` file if the user includes the `-j` or `--json-file` flag and the output file's path, e.g.:

```
deltacode -n [path to the 'new' codebase] -o [path to the 'old' codebase] -j [path to␣
↪the JSON output file]
```

Once a user has generated a DeltaCode JSON output file, he or she can convert that `JSON` output to `CSV` format by running a command with this structure::

```
python etc/scripts/json2csv.py [path to the JSON input file] [path to the CSV output␣
↪file]
```

See also *JSON to CSV Conversion*.

Overall Structure

JSON

Top-Level JSON

DeltaCode's `JSON` output comprises the following six fields/keys and values at the top level:

1. `deltacode_notice` – A string of the terms under which the DeltaCode output is provided.
2. `deltacode_options` – A JSON object containing three key/value pairs:
   - `--new` – A string identifying the path to the `JSON` file containing the ScanCode output of the codebase the user wants DeltaCode to treat as the 'new' codebase.
   - `--old` – A string identifying the path to the JSON file containing the ScanCode output of the codebase the user wants DeltaCode to treat as the 'old' codebase.
   - **`--all-delta-types` – A `true` or `false` value.**
     - This value will be true if the command-line input includes the `-a` or `--all-delta-types` flag, in which case the deltas field described below will include details for unmodified files as well as all changed files.
     - If the user does not include the `-a` or `--all-delta-types` flag, the value will be false and unmodified files will be omitted from the DeltaCode output.
3. `deltacode_version` – A string representing the version of DeltaCode on which the codebase comparison was run.
4. `deltacode_errors` – A list of one or more strings identifying errors (if any) that occurred during the codebase-comparison process.

5. `deltas_count` – An integer representing the number of 'Delta' objects – the file-level comparisons of the two codebases (discussed in the next section) – contained in the DeltaCode output's `deltas` key/value pair.

   - If the user's command-line input does not include the `-a` or `--all-delta-types` flag (see the discussion above of the `--all-delta-types` field/key), the DeltaCode output will omit details for unmodified files and consequently the deltas_count field will not include unmodified files.

6. `deltas` – A list of 'Delta' objects, each of which represents a file-level comparison (i.e., the "delta") of the 'new' and 'old' codebases. The Delta object is discussed in further detail in the next section.

This is the top-level `JSON` structure of the key/value pairs described above:

```
{
  "deltacode_notice": "",
  "deltacode_options": {
    "--new": "",
    "--old": "",
    "--all-delta-types": false
  },
  "deltacode_version": "",
  "deltacode_errors": [],
  "deltas_count": 0,
  "deltas": [one or more Delta objects]
}
```

The Delta Object

Each Delta object consists of four key/value pairs:

- `factors`: A list of one or more strings representing the factors that characterize the file-level comparison and are used to calculate the resulting score, e.g.

  ```
  "factors": [
        "added",
        "license info added",
        "copyright info added"
      ],
  ```

The possible values for the factors field are discussed in some detail in DeltaCode Scoring *Deltacode Scoring*.

- `score`: An integer representing the magnitude/importance of the file-level change – the higher the `score`, the greater the change. For further details about the DeltaCode scoring system, see DeltaCode Scoring *Deltacode Scoring*.

- `new`: A 'File' object containing key/value pairs of certain ScanCode-based file attributes (`path`, `licenses`, `copyrights` etc.) for the file in the codebase designated by the user as `new`. If the Delta object represents the removal of a file (the `factors` value would be `removed`), the value of `new` will be `null`.

- `old`: A 'File' object containing key/value pairs of certain ScanCode-based file attributes for the file in the codebase designated by the user as `old`. If the Delta object represents the addition of a file (the `factors` value would be `added`), the value of `old` will be `null`.

The JSON structure of a Delta object looks like this::

```
{
  "factors": [],
  "score": 0,
  "new": {
    "path": "",
    "type": "",
```

(continues on next page)

(continued from previous page)

```
    "name": "",
    "size": 0,
    "sha1": "",
    "original_path": "",
    "licenses": [],
    "copyrights": []
  },
  "old": {
    "path": "",
    "type": "",
    "name": "",
    "size": 0,
    "sha1": "",
    "original_path": "",
    "licenses": [],
    "copyrights": []
  }
}
```

The File Object

As you saw in the preceding section, the File object has the following JSON structure::

```
{
  "path": "",
  "type": "",
  "name": "",
  "size": 0,
  "sha1": "",
  "original_path": "",
  "licenses": [],
  "copyrights": []
}
```

A File object consists of eight key/value pairs:

- `path`: – A string identifying the path to the file in question. In processing the ‘new’ and ‘old’ codebases to be compared, DeltaCode may modify the codebases’ respective file paths in order to properly align them for comparison purposes. As a result, a File object’s `path` value may differ to some extent from its `original_path` value (see below).
- `type`: – A string indicating whether the object is a `file` or a `directory`.
- `name`: – A string reflecting the name of the file.
- `size`: – An integer reflecting the size of the file in KB.
- `sha1`: – A string reflecting the file’s sha1 value.
- `original_path`: – A string identifying the file’s path as it exists in the codebase, prior to any processing by DeltaCode to modify the path for purposes of comparing the two codebases.
- `licenses`: – A list of License objects reflecting all licenses identified by ScanCode as associated with the file. This list can be empty.
- `copyrights`: – A list of Copyright objects reflecting all copyrights identified by ScanCode as associated with the file. This list can be empty.

Example of Detailed JSON output

Here is an example of the detailed DeltaCode output in `JSON` format displaying one Delta object in the `deltas` key/value pair – in this case, an excerpt from the `JSON` output of a DeltaCode comparison of `zlib-1.2.11` and `zlib-1.2.9`::

```
{
  "deltacode_notice": "Generated with DeltaCode and provided on an \"AS IS\" BASIS,␣
↪WITHOUT WARRANTIES\nOR CONDITIONS OF ANY KIND, either express or implied. No␣
↪content created from\nDeltaCode should be considered or used as legal advice.␣
↪Consult an Attorney\nfor any legal advice.\nDeltaCode is a free software codebase-
↪comparison tool from nexB Inc. and others.\nVisit https://github.com/nexB/deltacode/
↪ for support and download.",
  "deltacode_options": {
    "--new": "C:/scans/zlib-1.2.11.json",
    "--old": "C:/scans/zlib-1.2.9.json",
    "--all-delta-types": false
  },
  "deltacode_version": "1.0.0.post49.e3ff7be",
  "deltacode_errors": [],
  "deltas_count": 40,
  "deltas": [
    {
      "factors": [
        "modified"
      ],
      "score": 20,
      "new": {
        "path": "trees.c",
        "type": "file",
        "name": "trees.c",
        "size": 43761,
        "sha1": "ab030a33e399e7284b9ddf9bba64d0dd2730b417",
        "original_path": "zlib-1.2.11/trees.c",
        "licenses": [
          {
            "key": "zlib",
            "score": 60.0,
            "short_name": "ZLIB License",
            "category": "Permissive",
            "owner": "zlib"
          }
        ],
        "copyrights": [
          {
            "statements": [
              "Copyright (c) 1995-2017 Jean-loup Gailly"
            ],
            "holders": [
              "Jean-loup Gailly"
            ]
          }
        ]
      },
      "old": {
        "path": "trees.c",
        "type": "file",
        "name": "trees.c",
        "size": 43774,
        "sha1": "1a554d4edfaecfd377c71b345adb647d15ff7221",
```

(continues on next page)

(continued from previous page)

```
        "original_path": "zlib-1.2.9/trees.c",
        "licenses": [
          {
            "key": "zlib",
            "score": 60.0,
            "short_name": "ZLIB License",
            "category": "Permissive",
            "owner": "zlib"
          }
        ],
        "copyrights": [
          {
            "statements": [
              "Copyright (c) 1995-2016 Jean-loup Gailly"
            ],
            "holders": [
              "Jean-loup Gailly"
            ]
          }
        ]
      }
    },
    [additional Delta objects if any]
  ]
}
```

CSV

Compared with DeltaCode's JSON output, the CSV output is relatively simple, comprising the following seven fields as column headers, with each row representing one Delta object:

- `Score` – An integer representing the magnitude/importance of the file-level change.
- `Factors` – One or more strings – with no comma or other separators – representing the factors that characterize the file-level comparison and are used to calculate the resulting score.
- `Path` – A string identifying the file's path in the 'new' codebase unless the Delta object reflects a `removed` file, in which case the string identifies the file's path in the 'old' codebase. As noted above, this path may vary to some extent from the file's actual path in its codebase as a result of DeltaCode processing for codebase comparison purposes.
- `Name` – A string reflecting the file's name in the 'new' codebase unless the Delta object reflects a `removed` file, in which case the string reflects the file's name in the 'old' codebase.
- `Type` – A string reflecting the file's type ('file' or 'directory') in the 'new' codebase unless the Delta object reflects a `removed` file, in which case the string reflects the file's type in the 'old' codebase.
- `Size` – An integer reflecting the file's size in KB in the 'new' codebase unless the Delta object reflects a `removed` file, in which case the string reflects the file's size in the 'old' codebase.
- `Old Path` – A string reflecting the file's path in the 'old' codebase if the Delta object reflects a `moved` file. If the Delta object does not involve a `moved` file, this field is empty. As with the `Path` field/column header above, this path may differ to some extent from the file's actual path in its codebase due to DeltaCode processing for codebase comparison purposes.

1.2.3 Deltacode Scoring

Delta Objects

A File-Level Comparison of Two Codebases

A Delta object represents the file-level comparison (i.e., the "delta") of two codebases, typically two versions of the same codebase, using ScanCode-generated `JSON` output files as input for the comparison process.

Based on how the user constructs the command-line input, DeltaCode's naming convention treats one codebase as the "new" codebase and the other as the "old" codebase::

```
deltacode -n [path to the 'new' codebase] -o [path to the 'old' codebase] [...]
```

Basic Scoring

A DeltaCode codebase comparison produces a collection of file-level Delta objects. Depending on the nature of the file-level change between the two codebases, each Delta object is characterized as belonging to one of the categories listed below. Each category has an associated score intended to convey its potential importance – from a license/copyright compliance perspective – to a user's analysis of the changes between the `new` and `old` codebases.

In descending order of importance, the categories are:

1. `added`: A file has been added to the `new` codebase.
2. `modified`: The file is contained in both the `new` and `old` codebase and has been modified (as reflected, among other things, by a change in the file's `sha1` attribute).
3. `moved`: The file is contained in both the `new` and `old` codebase and has been moved but not modified.
4. `removed`: A file has been removed from the `old` codebase.
5. `unmodified`: The file is contained in both the `new` and `old` codebase and has not been modified or moved.

Note: Files are determined to be Moved by looping thru the *added* and *removed* Delta objects and checking their sha1 values.

The score of a Delta object characterized as `added` or `modified` may be increased based on the detection of license- and/or copyright-related changes. See *License Additions and Changes* and *Copyright Holder Additions and Changes* below.

Delta Object Fields and Values

Each Delta object includes the following fields and values:

- `factors`: One or more strings representing the factors that characterize the file-level comparison and resulting score, e.g., in JSON format::

```
"factors": [
  "added",
  "license info added",
  "copyright info added"
],
```

- `score`: A number representing the magnitude/importance of the file-level change – the higher the score, the greater the change.
- `new`: The ScanCode-based file attributes (`path`, `licenses`, `copyrights` etc.) for the file in the codebase designated by the user as `new`.
- `old`: The ScanCode-based file attributes for the file in the codebase designated by the user as `old`.

Note that an `added` Delta object will have a `new` file but no `old` file, while a `removed` Delta object will have an `old` file but not a `new` file. In each case, the `new` and `old` keys will be present but the value for the missing file will be `null`.

License Additions and Changes

Certain file-level changes involving the license-related information in a Delta object will increase the object's score.

- An `added` Delta object's score will be increased:
  - If the `new` file contains one or more licenses (`factors` will include `license info added`).
  - If the the `new` file contains any of the following Commercial/Copyleft license categories (`factors` will include, e.g., `copyleft` added):
    - 'Commercial'
    - 'Copyleft'
    - 'Copyleft Limited'
    - 'Free Restricted'
    - 'Patent License'
    - 'Proprietary Free'
- A `modified` Delta object's score will be increased:
  - If the `old` file has at least one license and the `new` file has no licenses (`factors` will include `license info removed`).
  - If the `old` file has no licenses and the `new` file has at least one license (`factors` will include `license info added`).
  - If both the `old` file and `new` file have at least one license and the license keys are not identical (e.g., the `old` file includes an `mit` license and an `apache-2.0` license and the `new` file includes only an `mit` license) (`factors` will include `license` change).
  - If any of the Commercial/Copyleft license categories listed above are found in the `new` file but not in the `old` file (`factors` will include, e.g., `proprietary free added`).

Copyright Holder Additions and Changes

- An `added` Delta object's score will be increased if the `new` file contains one or more copyright `holders` (`factors` will include `copyright info added`).
- A `modified` Delta object's score will be increased:
  - If the `old` file has at least one copyright `holder` and the `new` file has no copyright holders (`factors` will include `copyright info removed`).
  - If the `old` file has no copyright `holders` and the `new` file has at least one (`actors` will include `copyright info added`).

- If both the `old` file and `new` file have at least one copyright `holder` and the `holders` are not identical (`factors` will include `copyright` change).

Moved, Removed and Unmodified

As noted above in Basic Scoring *Basic Scoring*, from a license/copyright compliance perspective, the three least significant Delta categories are `moved`, `removed` and `unmodified`.

In the current version of DeltaCode, each of these three categories is assigned a score of 0, with no options to increase that score depending on the content of the Delta object.

However, it is possible that both `moved` and `removed` will be assigned some non-zero score in a future version. In particular, `removed` could be significant from a compliance viewpoint where, for example, the removal of a file results in the removal of a Commercial/Copyleft license obligation.

1.2.4 Development

See CONTRIBUTING.rst for details: https://github.com/nexB/deltacode/blob/develop/CONTRIBUTING.rst

Code layout and conventions

Source code is in `src/`. Tests are in `tests/`.

Each test script is named `test_XXXX` and while we love to use `py.test` as a test runner, most tests have no dependencies on `py.test`, only on the `unittest` module (with the exception of some command line tests that depend on pytest monkeypatching capabilities.

When source or tests need data files, we store these in a `data` subdirectory.

We use PEP8 conventions with a relaxed line length that can be up to 90'ish characters long when needed to keep the code clear and readable.

We store pre-built **bundled** native binaries in `bin/` sub-directories of each `src/` packages. These binaries are organized by OS and architecture. This ensure that DeltaCode works out of the box either using a checkout or a download, without needing a compiler and toolchain to be installed.

We store **bundled** thirdparty components and libraries in the `thirdparty` directory. Python libraries are stored as wheels, eventually pre-built if the corresponding wheel is not available in the Pypi repository.

Some of these components may be advanced builds with bug fixes or advanced patches.

We write tests, a lot of tests, thousands of tests. Several tests are data-driven and use data files as test input and sometimes data files as test expectation (in this case using either JSON or YAML files). The tests should pass on Linux 64 bits, Windows 32 and 64 bits and on MacOSX 10.6.8 and up. We maintain two CI loops with Travis (Linux) at https://travis-ci.org/nexB/deltacode and Appveyor (Windows) at https://ci.appveyor.com/project/nexB/deltacode

When finding bugs or adding new features, we add tests. See existing test code for examples.

Running tests

DeltaCode comes with over 130 unit tests to ensure detection accuracy and stability across Linux, Windows and macOS OSes: we kinda love tests, do we?

We use pytest to run the tests: call the `py.test` script to run the whole test suite. This is installed by `pytest` which is bundled with a DeltaCode checkout and installed when you run `./configure`).

If you are running from a fresh git clone and you run `./configure` and then `source bin/activate` the `py.test` command will be available in your path.

Alternatively if you have already configured but are not in an activated "virtualenv" the `py.test` command is available under `<root of your checkout>/bin/py.test`

(Note: paths here are for POSIX, but mostly the same applies to Windows)

If you have a multiprocessor machine you might want to run the tests in parallel (and faster) For instance: `py.test -n4` runs the tests on 4 CPUs. We typically run the tests in verbose mode with `py.test -vvs -n4`

See also https://docs.pytest.org for details or use the `py.test -h` command to show the many other options available.

One useful option is to run a select subset of the test functions matching a pattern with the `-k` option for instance: `py.test -vvs -k tcpdump` would only run test functions that contain the string "tcpdump" in their name or their class name or module name .

Another useful option after a test run with some failures is to re-run only the failed tests with the `--lf` option for instance: `py.test -vvs --lf` would only run only test functions that failed in the previous run.

pip requirements and the configure script

DeltaCode use the `configure` and `configure.bat` (and `etc/configure.py` behind the scenes) scripts to install a virtualenv, install required packaged dependencies as pip requirements and more configure tasks such that DeltaCode can be installed in a self-contained way with **no network connectivity** required.

DeltaCode requirements and third-party Python libraries

In a somewhat unconventional way, all the required libraries are bundled aka. copied in the repo itself in the thirdparty/ directory. If DeltaCode were only a library it would not make sense. But its is first an application and having a well defined frozen set of dependent packages is important for an app.

The benefit of this approach (combined with the `configure` script) means that a mere checkout of the repository contains **everything** needed to run DeltaCode except for a Python interpreter.

Using DeltaCode as a Python library

(Coming Soon) DeltaCode can be used alright as a Python library and is available as as a Python wheel in Pypi and installed with `pip install deltacode`.

1.2.5 JSON to CSV Conversion

The default output format for a DeltaCode codebase comparison is JSON. If the `-j` or `--json-file` option is included in the `deltacode` command, the output will be written to a `.json` file at the user-designated location. For example:

```
deltacode -n [path to the 'new' codebase] -o [path to the 'old' codebase] -j [path to␣
↪the JSON output file]
```

We have also created an easy-to-use script for users who want to convert their JSON output to CSV format. Located at `etc/scripts/json2csv.py`, the conversion can be run with this command template:

```
python etc/scripts/json2csv.py [path to the JSON input file] [path to the CSV output␣
↪file]
```

1.2.6 Release Process

Steps to cut a new release:

run bumpversion with major, minor or patch to bump the version in:

```
src/deltacode/__init__.py
setup.py
deltacode.ABOUT
```

Update the CHANGELOG.rst commit changes and push changes to develop:

```
git commit -m "commit message"
git push --set-upstream origin develop
```

merge develop branch in master and tag the release.

```
git checkout master
git merge develop
git tag -a v1.6.1 -m "Release v1.6.1"
git push --set-upstream origin master
git push --set-upstream origin v1.6.1
```

Draft a new release in GitHub, using the previous release blurb as a base. Highlight new and noteworthy changes from the CHANGELOG.rst.

Run etc/release/release.sh locally.

Upload the release archives created in the dist/ directory to the GitHub release page.

Save the release as a draft. Use the previous release notes to create notes in the same style. Ensure that the link to thirdparty source code is present.

Test the downloads.

Publish the release on GitHub

Then build and publish the released wheel on Pypi. For this you need your own Pypi credentials (and get authorized to publish Pypi release: ask @pombredanne) and you need to have the twine package installed and configured.

Build a .whl with `python setup.py bdist_wheel` Run twine with `twine upload dist/<path to the built wheel>` Once uploaded check the published release at https://pypi.python.org/pypi/deltacode/ Then create a new fresh local virtualenv and test the wheel installation with: `pip install deltacode`

1.2.7 Google Summer of Code 2019 - Final report

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, @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.

1.3 AboutCode Docs

Welcome to the AboutCode wiki!

If you are interested in the Google Summer of Code 2019, check out this page. *Google Summer of Code 2019*

If you are interested in the Google Season of Documents 2019, go through this page. *Google Season of Docs 2019*

1.3.1 Contributor Project Ideas

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

   - *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 : 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**
  - @majurg https://github.com/majurg
  - @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**
  - @majurg https://github.com/majurg
  - @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
  - @majurg https://github.com/majurg

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

1.3.2 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

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

1.3.3 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** :
  - @majurg https://github.com/majurg
  - @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** :
    - @majurg https://github.com/majurg
    - @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
    - @majurg https://github.com/majurg

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** :
  - @majurg https://github.com/majurg

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

1.3.4 Google Summer of Code 2018

See *Contributor Project Ideas*.

1.3.5 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**
  - @majurg https://github.com/majurg

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**
  - @majurg https://github.com/majurg
  - @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**
  - @majurg https://github.com/majurg
  - @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.

1.3.6 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>
```

1.4 AboutCode-Toolkit Documentation

1.4.1 AboutCode Toolkit

Build and tests status

Branch	**Linux/macOS**	**Windows**
Master	build passing	BUILD PASSING
Develop	build passing	BUILD PASSING

The AboutCode Toolkit and ABOUT files provide a simple way to document the origin, license, usage and other important or interesting information about third-party software components that you use in your project.

You start by storing ABOUT files (a small YAML formatted text file with field/value pairs) side-by-side with each of the third-party software components you use. Each ABOUT file documents origin and license for one software. For more information on the ABOUT file format, visit http://www.dejacode.org There are many examples of ABOUT files (valid or invalid) in the testdata/ directory of the whole repository.

The current version of the AboutCode Toolkit can read these ABOUT files so that you can collect and validate the inventory of third-party components that you use.

In addition, this tool is able to generate attribution notices and identify redistributable source code used in your project to help you comply with open source licenses conditions.

This version of the AboutCode Toolkit follows the ABOUT specification version 3.0 at: https://github.com/nexB/aboutcode-toolkit/blob/develop/SPECIFICATION.rst

REQUIREMENTS

The AboutCode Toolkit is tested with Python 2.7 and 3.6 on Linux, Mac and Windows. You will need to install a Python interpreter if you do not have one already installed.

On Linux and Mac, Python is typically pre-installed. To verify which version may be pre-installed, open a terminal and type:

> python –version

On Windows or Mac, you can download the latest Python here: https://www.python.org/downloads/

Download the .msi installer for Windows or the .dmg archive for Mac. Open and run the installer using all the default options.

INSTALLATION

Checkout or download and extract the AboutCode Toolkit from: https://github.com/nexB/aboutcode-toolkit/

To install all the needed dependencies in a virtualenv, run (on posix): source configure

or on windows: configure

REFERENCE

See https://github.com/nexB/aboutcode-toolkit/blob/master/REFERENCE.rst for reference on aboutcode-toolkit usage.

TESTS and DEVELOPMENT

To install all the needed development dependencies, run (on posix): source configure etc/conf/dev

or on windows: configure etc/conf/dev

To verify that everything works fine you can run the test suite with: py.test

HELP and SUPPORT

If you have a question or find a bug, enter a ticket at:

> https://github.com/nexB/aboutcode-toolkit

For issues, you can use:

> https://github.com/nexB/aboutcode-toolkit/issues

SOURCE CODE

The AboutCode Toolkit is available through GitHub. For the latest version visit: https://github.com/nexB/aboutcode-toolkit

HACKING

We accept pull requests provided under the same license as this tool. You agree to the http://developercertificate.org/

LICENSE

The AboutCode Toolkit is released under the Apache 2.0 license. See (of course) the about.ABOUT file for details.

1.4.2 Reference

about

Syntax

```
about [OPTIONS] [COMMANDS]
```

Options:

```
--version    Show the version and exit.
--help       Show this message and exit.
```

Commands:

```
attrib     LOCATION: directory, OUTPUT: output file
check      LOCATION: directory
gen        LOCATION: input file, OUTPUT: directory
inventory  LOCATION: directory, OUTPUT: csv file
```

attrib

Syntax

```
about attrib [OPTIONS] LOCATION OUTPUT

LOCATION: Path to an ABOUT file or a directory containing ABOUT files.
OUTPUT: Path to output file to write the attribution to.
```

Options:

```
--inventory PATH            Path to an inventory file.
--mapping                   Use for mapping between the input keys and the ABOUT␣
↪field.
                            names - mapping.config
--mapping-file              Use a custom mapping file with mapping between input
                            keys and ABOUT field names.
--template PATH             Path to a custom attribution template.
--vartext TEXT              Variable texts to the attribution template
--verbose                   Show all the errors and warning.
-q, --quiet                 Do not print any error/warning.
-h, --help                  Show this message and exit.
```

Purpose

Generate an attribution file which contains the all license information from the LOCATION along with the license text.

Assume the following:

```
'/home/about_files/'** contains all the ABOUT files [LOCATION]
'/home/attribution/attribution.html' is the user's output path [OUTPUT]
'/home/project/component_list.csv' is the inventory that user want to be generated
```

```
$ about attrib /home/about_files/ /home/attribution/attribution.html
```

Options

```
--inventory

    This option allows you to define which ABOUT files should be used for attribution␣
↪generation.
    For instance,
    '/home/project/component_list.csv' is the inventory that user want to be generated

$ about attrib --inventory /home/project/component_list.csv LOCATION OUTPUT

--mapping

    See mapping.config for details

--mapping-file

    Same behavior as `--mapping` but with custom mapping file

$ about attrib --mapping-file CUSTOM_MAPPING_FILE_PATH LOCATION OUTPUT

--template

    This option allows you to use your own template for attribution generation.
    For instance, if you have a custom template located at:
    /home/custom_template/template.html

$ about attrib --template /home/custom_template/template.html LOCATION OUTPUT

--vartext

    This option allow you to pass variable texts to the attribution template

$ about attrib --vartext "title=Attribution Notice" --vartext "header=Product 101"␣
↪LOCATION OUTPUT

    Users can use the following in the template to get the vartext:
    {{ vartext_dict['title'] }}
    {{ vartext_dict['header'] }}

--verbose

    This option tells the tool to show all errors found.
    The default behavior will only show 'CRITICAL', 'ERROR', and 'WARNING'
```

The following data are passed to jinja2 and, therefore, can be used for a custom template:

- about object: the about objects
- common_licenses: a common license keys list in licenses.py

- license_key_and_context: a dictionary list with license_key as a key and license text as the value
- license_file_name_and_key: a dictionary list with license file name as a key and license key as the value
- license_key_to_license_name: a dictionary list with license key as a key and license file name as the value

check

Syntax

```
about check [OPTIONS] LOCATION

LOCATION: Path to an ABOUT file or a directory with ABOUT files.
```

Options:

```
--verbose                 Show all the errors and warning
-h, --help                Show this message and exit.
```

Purpose

Validating ABOUT files at LOCATION.

Options

```
--verbose

    This option tells the tool to show all errors found.
    The default behavior will only show 'CRITICAL', 'ERROR', and 'WARNING'

$ about check --verbose /home/project/about_files/
```

gen

Syntax

```
about gen [OPTIONS] LOCATION OUTPUT

LOCATION: Path to a JSON or CSV inventory file.
OUTPUT: Path to a directory where ABOUT files are generated.
```

Options:

```
--fetch-license KEY                  Fetch licenses text from a DejaCode API. and
                                     create <license>.LICENSE side-by-side
                                     with the generated .ABOUT file using data
                                     fetched from a DejaCode License Library. The
                                     following additional options are required:

                                     api_url - URL to the DejaCode License Library
                                     API endpoint

```

(continues on next page)

(continued from previous page)

```
                                     api_key - DejaCode API key
                                     Example syntax:

                                     about gen --fetch-license 'api_url' 'api_key'
--license-notice-text-location PATH Copy the 'license_file' from the directory to
                                     the generated location.
--mapping                            Use for mapping between the input keys and
                                     the ABOUT field names - mapping.config
--mapping-file                       Use a custom mapping file with mapping between␣
↪input
                                     keys and ABOUT field names.
--verbose                            Show all the errors and warning.
-q, --quiet                          Do not print any error/warning.
-h, --help                           Show this message and exit.
```

Purpose

Given an inventory of ABOUT files at location, generate ABOUT files in base directory.

Options

```
--fetch-license

    Fetch licenses text from a DejaCode API. and create <license>.LICENSE side-by-side
    with the generated .ABOUT file using data fetched from a DejaCode License Library.

    This option requires 2 parameters:
        api_url - URL to the DJE License Library
        api_key - Hash key to authenticate yourself in the API.

    In addition, the input needs to have the 'license_expression' field.
    (Please contact nexB to get the api_* value to use for this feature)

$ about gen --fetch-license 'api_url' 'api_key' LOCATION OUTPUT

--license-notice-text-location

    Copy the license files and notice files to the generated location based on the
    'license_file' and 'notice_file' value in the input from the directory

    For instance,
    the directory, /home/licenses_notices/, contains all the licenses and notices␣
↪that you want:
    /home/license/apache2.LICENSE
    /home/license/jquery.js.NOTICE

$ about gen --license-notice-text-location /home/licenses_notices/ LOCATION OUTPUT

--mapping

    See mapping.config for details

--mapping-file
```

(continues on next page)

(continued from previous page)

```
    Same behavior as `--mapping` but with custom mapping file

$ about attrib --mapping-file CUSTOM_MAPPING_FILE_PATH LOCATION OUTPUT

--verbose

    This option tells the tool to show all errors found.
    The default behavior will only show 'CRITICAL', 'ERROR', and 'WARNING'
```

inventory

Syntax

```
about inventory [OPTIONS] LOCATION OUTPUT

LOCATION: Path to an ABOUT file or a directory with ABOUT files.
OUTPUT: Path to the JSON or CSV inventory file to create.
```

Options:

```
--filter TEXT                Filter for the output inventory.
-f, --format [json|csv]      Set OUTPUT file format.  [default: csv]
--mapping                    Use file mapping.config to collect the defined not␣
↪supported fields in ABOUT files.
--mapping-file               Use a custom mapping file with mapping between input
                             keys and ABOUT field names.
--mapping-output FILE        Use a custom mapping file with mapping between
                             ABOUT field names and output keys
--verbose                    Show all the errors and warning.
-q, --quiet                  Do not print any error/warning.
-h, --help                   Show this message and exit.
```

Purpose

Collect a JSON or CSV inventory of components from ABOUT files.

Options

```
-filter TEXT

    Filter for the output inventory.

$ about inventory --filter "license_expression=gpl-2.0" LOCATION OUTPUT

The above command will only inventory the ABOUT files which have the "license_
↪expression: gpl-2.0"

-f, --format [json|csv]

    Set OUTPUT file format.  [default: csv]
```

(continues on next page)

(continued from previous page)

```
$ about inventory -f json LOCATION OUTPUT

--mapping

    See mapping.config for details

--mapping-file

    Same behavior as `--mapping` but with custom mapping file

$ about inventory --mapping-file CUSTOM_MAPPING_FILE_PATH LOCATION OUTPUT

--mapping-output

    Same behavior as `--mapping-file` but with custom mapping file
    In the custom mapping file, the left side is the custom key name where
    the right side is the ABOUT field name. For instance,
    Component: name

    The "Component" is a custom field name for the output
    The "name" is one of the defaul ABOUT field name that user want to convert

$ about inventory --mapping-output CUSTOM_MAPPING_FILE_PATH LOCATION OUTPUT

--verbose

    This option tells the tool to show all errors found.
    The default behavior will only show 'CRITICAL', 'ERROR', and 'WARNING'
```

Special Notes

Multiple licenses support format

The multiple licenses support format for CSV files are separated by line break

about_resource	name	license_key	license_file
test.tar.xz	test	apache-2.0 mit	apache-2.0.LICENSE mit.LICENSE

The multiple licenses support format for ABOUT files are by “grouping” with the keyword “licenses”

```
about_resource: test.tar.xz
name: test
licenses:
    -   key: apache 2.0
        name: apache-2.0.LICENSE
    -   key: mit
        name: mit.LICENSE
```

1.4.3 ABOUT File Specification v3.1.2

Purpose

An ABOUT file provides a simple way to document the provenance (origin and license) and other important or interesting information about a software component. An ABOUT file is a small YAML formatted text file stored in the codebase side-by-side with the software component file or archive that it documents. No modification of the documented software is needed.

The ABOUT format is plain text with field name/value pairs separated by a colon. It is easy to read and create by hand and is designed first for humans, rather than machines. The format is well-defined and structured just enough to make it easy to process with software as well. It contains enough information to fulfill key license requirements such as creating credits or attribution notices, collecting redistributable source code, or providing information about new versions of a software component.

Getting Started

A simple and valid ABOUT file named httpd.ABOUT may look like this:

```
about_resource: httpd-2.4.3.tar.gz
name: Apache HTTP Server
version: 2.4.3
homepage_url: http://httpd.apache.org
download_url: http://archive.apache.org/dist/httpd/httpd-2.4.3.tar.gz
license_expression: apache-2.0
licenses:
    -   key: apache-2.0
    -   file: apache-2.0.LICENSE
notice_file: httpd.NOTICE
copyright: Copyright (c) 2012 The Apache Software Foundation.
```

The meaning of this ABOUT file is:

- The file "httpd-2.4.3.tar.gz" is stored in the same directory and side-by-side with the ABOUT file "httpd.ABOUT" that documents it.
- The name of this component is "Apache HTTP Server" with version "2.4.3".
- The home URL for this component is http://httpd.apache.org
- The file "httpd-2.4.3.tar.gz" was originally downloaded from `http://archive.apache.org/dist/httpd/httpd-2.4.3.tar.gz`
- In the same directory, "apache-2.0.LICENSE" and "httpd.NOTICE" are files that contain respectively the license text and the notice text for this component.
- This component is licensed under "apache-2.0"

Specification

An ABOUT file is an ASCII YAML formatted text file. Note that while Unicode characters are not supported in an ABOUT file proper, external files can contain UTF-8 Unicode.

ABOUT file name

An ABOUT file name can use a limited set of characters and is suffixed with a ".ABOUT" extension using any combination of uppercase and lowercase characters.

A file name can contain only these US-ASCII characters:

- digits from 0 to 9
- uppercase and lowercase letters from A to Z
- the following symbols: "_", "-", "+", ".", "(", ")", "~", "[", "]", "{", "}"
- The case of a file name is not significant. On case-sensitive file systems (such as on Linux), a tool must report an error if two ABOUT files stored in the same directory have the same lowercase file name. This is to ensure that ABOUT files can be used across file systems. The convention is to use a lowercase file name and an uppercase ABOUT extension.

Lines of text

An ABOUT file contains lines of US-ASCII text. Lines contain field names/values pairs. The standard line ending is the LF character. The line ending characters can be any LF, CR or CR/LF and tools must normalize line endings to LF when processing an ABOUT file. Empty lines and lines containing only white spaces that are not part of a field value continuation are ignored. Empty lines are commonly used to improve the readability of an ABOUT file.

Field name

A field name can contain only these US-ASCII characters:

- digits from 0 to 9
- uppercase and lowercase letters from A to Z
- the "_" underscore sign.
- Field names are not case sensitive. For example, "HOMEPAGE_URL" and "HomePage_url" represent the same field name.
- A field name must start at the beginning of a new line. It can be followed by one or more spaces that must be ignored. These spaces are commonly used to improve the readability of an ABOUT file.

Field value

The field value is separated from the field name by a ":" colon. The ":" colon can be followed by one or more spaces that must be ignored. This also applies to trailing white spaces: they must be ignored.

The field value is composed of one or more lines of plain US-ASCII printable text.

When a field value is a long string, additional continuation lines must start with at least one space. In this case, the first space of an additional continuation line is ignored and should be removed from the field value by tools.

For instance:

```
description: This is a long description for a
 software component that additional continuation line is used.


  When a field value contains more than one line of text,  a 'literal block'
  (using |) is need.
```

For instance:

```
description: |
    This is a long description for a software component that spans
    multiple lines with arbitrary line breaks.

    This text contains multiple lines.
```

Fields are mandatory or optional

As defined in this specification, a field can be mandatory or optional. Tools must report an error for missing mandatory fields.

Extension and ignored fields

An ignored field is a field with a name that is not defined in this specification. Custom extension fields are also supported and must be processed by tools as ignored fields unless a certain tool can process a certain extension field.

Fields validation

When processing an ABOUT file, tools must report a warning or error if a field is invalid. A field can be invalid for several reasons, such as invalid field name syntax or invalid content. Tools should report additional validation error details. The validation process should check that each field name is syntactically correct and that fields contain correct values according to its concise, common sense definition in this specification. For certain fields, additional and specific validations are relevant such as checksum verification, URL validation, path resolution and verification, and so forth. Tools should report a warning for ignored fields.

Fields order and multiple occurrences

The field order does not matter. Multiple occurrences of a field name is not supported.

The tool processing an ABOUT file or CSV/JSON input will issue an error when a field name occurs more than once in the input file (as for any other ignored field).

Field referencing a file

The actual value of some fields may be contained in another file. This is useful for long texts or to reference a common text in multiple ABOUT files such as a common license text. In this case the field name is suffixed with "_file" and the field value must be a path pointing to the file that contains the actual value of the field. This path must be a POSIX path relative to the path of the ABOUT file. The file content must be UTF-8-encoded text. This is in contrast with field values contained directly in an ABOUT file that must be US-ASCII- encoded text and allows to support non-ASCII text content.

For example, the full license text for a component is often stored in a separate file named COPYING:

```
licenses:
    -   file: linux.COPYING
```

In this example, the README file is stored in a doc directory, one directory above the ABOUT file directory, using a relative POSIX path:

```
licenses:
    -   file: ../docs/ruby.README
```

Field referencing a URL

The value of a field may reference URLs such as a homepage or a download. In this case the field name is suffixed with "_url" and the field value must be a valid absolute URL starting with `ftp://`, `http://` or `https://`. URLs are informational and the content they may reference is ignored. For example, a download URL is referenced this way:

```
download_url: http://www.kernel.org/pub/linux/kernel/v3.0/linux-3.4.20.tar.bz2
```

Flag fields

Flag fields have a "true" or "false" value. True, T, Yes or Y , x must be interpreted as "true" in any case combination. False, F, No or N must be interpreted as "false" in any case combination.

Referencing the file or directory documented by an ABOUT file

An ABOUT file documents one file or directory. The mandatory "about_resource" field reference the documented file or directory. The value of the "about_resource" field is the name or path of the referenced file or directory.

A tool processing an ABOUT file must report an error if this field is missing.

By convention, an ABOUT file is often stored in the same directory side-by-side to the file or directory that it documents, but this is not mandatory.

For example, a file named django.ABOUT contains the following field to document the django-1.2.3.tar.gz archive stored in the same directory:

```
about_resource: django-1.2.3.tar.gz
```

In this example, the ABOUT file documents a whole sub-directory:

```
about_resource: linux-kernel-2.6.23
```

In this example, the ABOUT file documents the current directory, using a "." period to reference it:

```
about_resource: .
```

Other Mandatory fields

When a tool processes an ABOUT file, it must issue an error if these mandatory field are missing.

- about_resource: The resource this file referencing to.
- name: Component name.

Optional Information fields

- version: Component or package version. A component or package usually has a version, such as a revision number or hash from a version control system (for a snapshot checked out from VCS such as Subversion or Git). If not available, the version should be the date the component was provisioned, in an ISO date format such as 'YYYY-MM-DD'.
- spec_version: The version of the ABOUT file format specification used for this file. This is provided as a hint to readers and tools in order to support future versions of this specification.
- description: Component description, as a short text.
- download_url: A direct URL to download the original file or archive documented by this ABOUT file.
- homepage_url: URL to the homepage for this component.
- changelog_file: Changelog file for the component.
- notes: Notes and comments about the component.

Optional Owner and Author fields

- owner: The name of the primary organization or person(s) that owns or provides the component.
- owner_url: URL to the homepage for the owner.
- contact: Contact information (such as an email address or physical address) for the component owner.
- author: Name of the organization(s) or person(s) that authored the component.
- author_file: Author file for the component.

Optional Licensing fields

- copyright: Copyright statement for the component.
- notice_file: Legal notice or credits for the component.
- notice_url: URL to a legal notice for the component.
- license_file: License file that applies to the component. For example, the name of a license file such as LICENSE or COPYING file extracted from a downloaded archive.
- license_url: URL to the license text for the component.
- license_expression: The license expression that apply to the component. You can separate each identifier using " or " and " and " to document the relationship between multiple license identifiers, such as a choice among multiple licenses.
- license_name: The license short name for the license.
- license_key: The license key(s) for the component.

Optional Boolean flag fields

- redistribute: Set this flag to yes if the component license requires source code redistribution. Defaults to no when absent.
- attribute: Set this flag to yes if the component license requires publishing an attribution or credit notice. Defaults to no when absent.

- track_changes: Set this flag to yes if the component license requires tracking changes made to a the component. Defaults to no when absent.
- modified: Set this flag to yes if the component has been modified. Defaults to no when absent.
- internal_use_only: Set this flag to yes if the component is used internal only. Defaults to no when absent.

Optional Extension fields

You can create extension fields by prefixing them with a short prefix to distinguish these from the standard fields. You should provide documentation for these extensions and create or extend existing tools to support these extensions. Other tools must ignore these extensions.

Optional Extension fields to reference files stored in a version control system (VCS)

These fields provide a simple way to reference files stored in a version control system. There are many VCS tools such as CVS, Subversion, Git, ClearCase and GNU Arch. Accurate addressing of a file or directory revision in each tool in a uniform way may not be possible. Some tools may require access control via user/password or certificate and this information should not be stored in an ABOUT file. This extension defines the 'vcs' field extension prefix and a few common fields to handle the diversity of ways that VCS tools reference files and directories under version control:

- vcs_tool: VCS tool such as git, svn, cvs, etc.
- vcs_repository: Typically a URL or some other identifier used by a VCS tool to point to a repository such as an SVN or Git repository URL.
- vcs_path: Path used by a particular VCS tool to point to a file, directory or module inside a repository.
- vcs_tag: tag name or path used by a particular VCS tool.
- vcs_branch: branch name or path used by a particular VCS tool.
- vcs_revision: revision identifier such as a revision hash or version number.

Some examples for using the vcs_* extension fields include:

```
vcs_tool: svn
vcs_repository: http://svn.code.sf.net/p/inkscape/code/inkscape_project/
vcs_path: trunk/inkscape_planet/
vcs_revision: 22886
```

or:

```
vcs_tool: git
vcs_repository: git://git.kernel.org/pub/scm/linux/kernel/git/stable/linux-stable.git
vcs_path: tools/lib/traceevent
vcs_revision: b59958d90b3e75a3b66cd311661535f94f5be4d1
```

Optional Extension fields for checksums

These fields support checksums (such as SHA1 and MD5)commonly provided with downloaded archives to verify their integrity. A tool can optionally use these to verify the integrity of a file documented by an ABOUT file.

- checksum_md5: MD5 for the file documented by this ABOUT file in the "about_resource" field.
- checksum_sha1: SHA1 for the file documented by this ABOUT file in the "about_resource" field.

Some examples:

```
checksum_md5: f30b9c173b1f19cf42ffa44f78e4b96c
```

1.5 AboutCode Data : ABCD

1.5.1 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.

1.5.2 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.

1.5.3 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.

1.5.4 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.

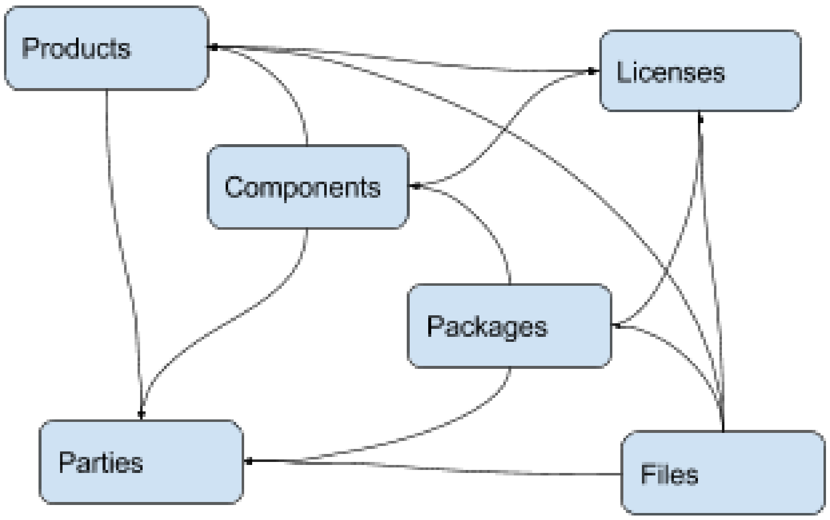
And these secondary, important but less prominent object types:

- 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).

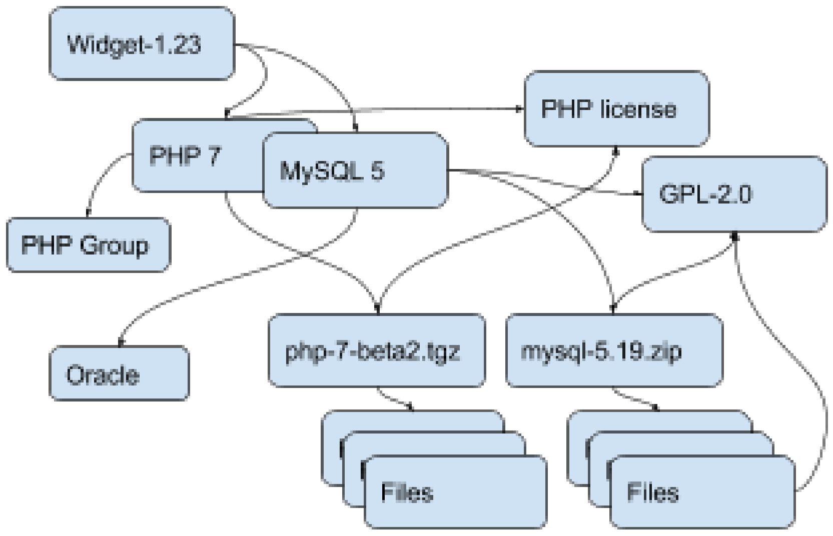
Each of these objects has a few identifying attributes and eventually many tool- or application-specific data attributes. Each tool defines and documents the attributes they can handle and care for. When some agreement is reached on the definition of new attributes or objects, the ABCD dictionary may 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.

1.5.5 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.

1.5.6 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.

1.5.7 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.

1.5.8 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):

```
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.

1.5.9 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*:

```
{"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"
        }
```

(continues on next page)

(continued from previous page)

```
    ]
}]}
```

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:

```
{"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:

```
"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 embedded 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"},
    ]
}]
```

1.5.10 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.

1.5.11 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.

1.5.12 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.

1.5.13 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.

1.5.14 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.

1.5.15 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\\/"
```

1.5.16 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.

1.5.17 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:

```
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

    license_expression: psf

    licenses:
      - source: scancode.com
        key: psf
        text_file: PSF.LICENSE
```

And here would be similar data in 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",
```

(continues on next page)

(continued from previous page)

```
               "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:

```
{
  "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",
```

(continues on next page)

(continued from previous page)

```
      "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.",
```

(continues on next page)

(continued from previous page)

```
        "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": [
```

(continues on next page)

(continued from previous page)

```
        {
          "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.

CHAPTER 2

Documentation Guide

2.1 Help

AboutCode is a suite of open source projects.

2.1.1 AboutCode Projects

- ScanCode Toolkit: This is a set of code scanning tools to detect the origin and license of code and dependencies. ScanCode Toolkit uses a plug-in architecture to run a series of scan-related tools in one process flow. This is the most popular project and is used by hundreds of software teams. https://github.com/nexB/scancode-toolkit . The lead maintainer is @pombredanne
- Scancode Workbench (formerly AboutCode Manager) This 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. https://github.com/nexB/aboutcode-manager . The lead maintainer is @majurg
- AboutCode Toolkit: This is a set of command line tools to document the provenance of your code and generate attribution notices. AboutCode Toolkit uses small yaml files to document code provenance inside a codebase. https://github.com/nexB/aboutcode-toolkit . The lead maintainer is @chinyeungli
- TraceCode Toolkit: This is a set of tools to trace files from your deployment or distribution packages back to their origin in a development codebase or repository. The primary tool uses strace https://github.com/strace/strace/ to trace system calls on Linux and construct a build graph from syscalls to show which files are used to build a binary. We are contributors to strace. Maintained by @pombredanne
- Conan: "conan" stands for "CONtainer ANalysis" and is a tool to analyze the structure and provenance of software components in Docker images using static analysis. https://github.com/nexB/conan Maintained by @pombredanne
- license-expression: This is a library to parse, analyze, compare and normalize SPDX-like license expressions using a boolean logic expression engine. See https://spdx.org/spdx-specification-21-web-version#h.jxpfx0ykyb60 to understand what a license expression is. See https://github.com/nexB/license-expression for the code. The underlying boolean engine is at https://github.com/bastikr/boolean.py . Both are co-maintained by @pombredanne

- ABCD aka AboutCode Data: is a simple set of conventions to define data structures that all the AboutCode tools can understand and use to exchange data. The specification lives in this repository. .ABOUT files and ScanCode tooklit data are examples of this approach. Other projects such as https://libraries.io and OSS Review Toolkit also use these conventions.
- DeltaCode is a command line tool to compare scans and determine if and where there are material differences that affect licensing. The lead maintainer is @majurg
- VulnerableCode: an emerging server-side application to collect and track known package vulnerabilities.

2.2 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/

2.3 Document Maintenance

2.3.1 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

2.3.2 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 bin/activate
```

Now you can install Sphinx and the format theme used by readthedocs:

```
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.

2.3.3 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
```

2.3.4 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. You can review your results online.

2.3.5 Documentation Style Guides

The `scancode-toolkit` documentation is compliant to our doc style standards, enforced using `doc8`. For more details refer contrib_doc_dev.

CHAPTER 3

Indices and Tables

- genindex
- modindex