

---

# aboutcode-toolkit

nexb Inc.

Mar 04, 2024

# CONTENTS

<b>1</b>	<b>AboutCode Toolkit</b>	<b>2</b>
1.1	Introduction . . . . .	2
1.2	Build and tests status . . . . .	2
1.3	REQUIREMENTS . . . . .	2
1.4	INSTALLATION . . . . .	3
1.5	ACTIVATE the VIRTUALENV . . . . .	3
1.6	DEACTIVATE the VIRTUALENV . . . . .	3
1.7	VERSIONING SCHEMA . . . . .	3
1.8	REFERENCE . . . . .	4
1.9	TESTS and DEVELOPMENT . . . . .	4
1.10	CLEAN BUILD AND INSTALLED FILES . . . . .	4
1.11	HELP and SUPPORT . . . . .	4
1.12	SOURCE CODE . . . . .	4
1.13	HACKING . . . . .	5
1.14	LICENSE . . . . .	5
<b>2</b>	<b>General</b>	<b>6</b>
2.1	AboutCode Toolkit Defined . . . . .	6
2.2	Key Terminology . . . . .	7
2.3	Using gen to Generate ABOUT file(s) . . . . .	7
2.4	Using attrib to Generate a Product Attribution Notice Package . . . . .	10
2.5	Using inventory to Generate a Software Inventory . . . . .	13
<b>3</b>	<b>ABOUT File Specification v3.3.2</b>	<b>14</b>
3.1	Purpose . . . . .	14
3.2	Getting Started . . . . .	14
3.3	Specification . . . . .	15
<b>4</b>	<b>Reference</b>	<b>22</b>
4.1	about . . . . .	22
4.2	attrib . . . . .	23
4.3	check . . . . .	25
4.4	collect_redist_src . . . . .	27
4.5	gen . . . . .	28
4.6	gen_license . . . . .	31
4.7	inventory . . . . .	32
4.8	transform . . . . .	34
4.9	Notes . . . . .	38
<b>5</b>	<b>Type of Errors</b>	<b>39</b>

5.1	NOTSET . . . . .	39
5.2	DEBUG . . . . .	39
5.3	INFO . . . . .	39
5.4	WARNING . . . . .	40
5.5	ERROR . . . . .	40
5.6	CRITICAL . . . . .	40

Welcome to the AboutCode Toolkit's Documentation.

## ABOUTCODE TOOLKIT

### 1.1 Introduction

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. 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 3.3.2 at: <https://aboutcode-toolkit.readthedocs.io/en/latest/specification.html>

### 1.2 Build and tests status

Branch	Linux/macOS	Windows
Master		 BUILD PASSING
Develop		 BUILDING...

### 1.3 REQUIREMENTS

The AboutCode Toolkit is tested with Python 3.7 or above only 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
```

---

**Note:** Debian has decided that `distutils` is not a core python package, so it is not included in the last versions of debian and debian-based OSes.

A solution is to run: `sudo apt install python3-distutils`

---

**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.

## 1.4 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):**

`./configure`

**or on windows:**

`configure`

## 1.5 ACTIVATE the VIRTUALENV

**To activate the virtualenv, run (on posix):**

`source venv/bin/activate`

**or on windows:**

`venv\bin\activate`

## 1.6 DEACTIVATE the VIRTUALENV

**To deactivate the virtualenv, run (on both posix and windows):**

`deactivate`

## 1.7 VERSIONING SCHEMA

Starting at AboutCode version 4.0.0, the AboutCode Toolkit will follow SemVer for the versioning schema.

**i.e. MAJOR.MINOR.PATCH format**

1. MAJOR version when making incompatible API changes,
2. MINOR version when making functionality in a backwards compatible manner, and
3. PATCH version when making backwards compatible bug fixes.

## 1.8 REFERENCE

See <https://aboutcode-toolkit.readthedocs.io/en/latest/> for documentation.

See <https://aboutcode-toolkit.readthedocs.io/en/latest/reference.html> for reference.

## 1.9 TESTS and DEVELOPMENT

**To install all the needed development dependencies, run (on posix):**

```
./configure --dev
```

**or on windows:**

```
configure --dev
```

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

```
pytest
```

## 1.10 CLEAN BUILD AND INSTALLED FILES

**To clean the built and installed files, run (on posix):**

```
./configure --clean
```

**or on windows:**

```
configure --clean
```

## 1.11 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>

## 1.12 SOURCE CODE

**The AboutCode Toolkit is available through GitHub. For the latest version visit:**

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

## 1.13 HACKING

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

## 1.14 LICENSE

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



## 2.1 AboutCode Toolkit Defined

AboutCode Toolkit is a tool for your software development team to document your code inside your codebase, typically in preparation for a product release, side-by-side with the actual code. ABOUT file(s) have a simple, standard format that identifies components and their associated licenses. The current AboutCode Toolkit subcommands are:

- **attrib**: Generate a Product Attribution notice document from your ABOUT file(s), JSON, CSV or XLSX. You can also generate documents for other purposes (such as a License Reference) by varying your input control file and your template.
- **check**: A simple command to validate the ABOUT file(s) and output errors/warnings on the terminal.
- **collect\_redist\_src**: A command to collect and copy sources that have the ‘redistribute’ flagged as ‘True’ in ABOUT file(s) or from an inventory.
- **gen**: Create ABOUT file(s) from a Software Inventory file (.csv, .json or .xlsx format) which is typically created from a software audit, and insert these AboutCode Toolkit files into your codebase. You can regenerate the AboutCode Toolkit files from a new Software Inventory file whenever you make changes.
- **gen\_license**: Fetch licenses in the license\_expression field and save to the output location.
- **inventory**: Generate a Software Inventory list (.csv, .json or .xlsx format) from your codebase based on ABOUT file(s). Note that this Software Inventory will only include components that have AboutCode Toolkit data. In another word, if you do not create AboutCode Toolkit files for your own original software components, these components will not show up in the generated inventory.
- **transform**: A command to transform an input CSV/JSON/XLSX by applying renaming and/or filtering and then output to a new CSV/JSON/XLSX file.

Additional AboutCode Toolkit information is available at:

- See *ABOUT File Specification v3.3.2* for an overview and a link to the ABOUT File specification.
- <https://github.com/nexB/aboutcode-toolkit/> for the AboutCode Toolkit tools.

## 2.2 Key Terminology

Some key terminology that applies to AboutCode Toolkit tool usage:

- **Software Inventory or Inventory** - means a list of all of the components in a Development codebase and the associated data about those components with a focus on software pedigree/provenance- related data for open source and third-party components.
- **Product BOM or BOM** - means a subset list of the components in a Development codebase (Software Inventory) that are Deployed on a particular Product Release (a Product Bill of Materials).

## 2.3 Using gen to Generate ABOUT file(s)

### 2.3.1 Prepare Your Software Inventory for gen Standard Field Names

You should start with a software inventory of your codebase in spreadsheet or JSON format. You need to prepare a version of it that will identify the field values that you want to appear in your .ABOUT files. Note the following standard field names (defined in the ABOUT File Specification), which gen will use to look for the values that it will store in your generated .ABOUT files, as well as any additional text files that you identify, which it will copy and store next to the .ABOUT files.

Standard Field Name	Description	Notes
about_resource_name	Name/path of the component resource	Mandatory
component_name	Component name	Mandatory
ignored_resources	List of paths ignored from the about_resource	Optional
version	Component version	Optional
download_url	Direct URL to download the original file or archive documented by this ABOUT file	Optional
description	Component description	Optional
homepage_url	URL to the homepage for this component	Optional
package_url	Package URL for this component (See <a href="https://github.com/package-url/purl-spec">https://github.com/package-url/purl-spec</a> for SPEC)	Optional
notes	notes text	Optional
license_expression	Expression for the license of the component using ScanCode license key(s).	Optional. 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_key	ScanCode license key for the component.	Optional. gen will obtain license information from ScanCode LicenseDB or DejaCode Enterprise if the <code>–fetch-license</code> or <code>–fetch-license-djc</code> option is set, including the license text, in order to create and write the appropriate .LICENSE file in the .ABOUT file target directory.
license_name	License name for the component.	Optional. This field will be generated if the <code>–fetch-license</code> or <code>–fetch-license-djc</code> option is set.

continues on next page

Table 1 – continued from previous page

Standard Field Name	Description	Notes
license file	license file name	Optional. gen will look for the file name (if a directory is specified in the <code>-reference</code> option) to copy that file to the <code>.ABOUT</code> file target directory.
license_url	URL to the license text for the component	Optional
spdx_license_key	The ScanCode LicenseDB <code>spdx_license_key</code> defined for the license at <a href="https://scancode-licensedb.aboutcode.org/index.html">https://scancode-licensedb.aboutcode.org/index.html</a>	Optional
copyright	copyright statement for the component	Optional
notice_file	notice text file name	Optional
notice_url	URL to the notice text for the component	Optional
redistribute	Yes/No. Does the component license require source redistribution.	Optional
attribution	Yes/No. Does the component license require publishing an attribution or credit notice.	Optional
track_changes	Yes/No. Does the component license require tracking changes made to the component.	Optional
modified	Yes/No. Have the component been modified.	Optional
internal_use_only	Yes/No. Is the component internal use only.	Optional
changelog	changelog text file name	Optional
owner	name of the organization or person that owns or provides the component	Optional
owner_url	URL to the owner for the component	Optional
contact	Contact information	Optional
author	author of the component	Optional
author_file	author text file name	Optional
vcs_tool	Name of the version control tool.	Optional
vcs_repository	Name of the version control repository.	Optional
vcs_path	Name of the version control path.	Optional
vcs_tag	Name of the version control tag.	Optional
vcs_branch	Name of the version control branch.	Optional
vcs_revision	Name of the version control revision.	Optional
checksum_md5	MD5 value for the file	Optional
checksum_sha1	SHA1 value for the file	Optional
checksum_sha256	SHA256 value for the file	Optional
spec_version	The version of the ABOUT file format specification used for this file.	Optional

## 2.3.2 Fields Renaming and Optional Custom Fields

Since your input's field name may not match with the AboutCode Toolkit standard field name, you can use the transform subcommand to do the transformation.

A transform configuration file is used to describe which transformations and validations to apply to a source CSV/JSON/XLSX file. This is a simple text file using YAML format, using the same format as an .ABOUT file.

The attributes that can be set in a configuration file are:

- **field\_renamings:** An optional map of source field name to target new field name that is used to rename CSV/JSON/XLSX fields.

```
field_renamings:
  about_resource : 'Directory/Location'
  bar : foo
```

The renaming is always applied first before other transforms and checks. All other field names referenced below are AFTER the renaming have been applied. For instance with this configuration, the field "Directory/Location" will be renamed to "about\_resource" and "foo" to "bar":

- **required\_fields:** An optional list of required field names that must have a value, beyond the standard field names. If a source CSV/JSON/XLSX does not have such a field or an entry is missing a value for a required field, an error is reported.

For instance with this configuration, an error will be reported if the fields "name" and "version" are missing, or if any entry does not have a value set for these fields:

```
required_fields:
- name
- version
```

- **field\_filters:** An optional list of fields that should be kept in the transformed file. If this list is provided, only the fields that are in the list will be kept. All others will be filtered out even if they are AboutCode Toolkit standard fields. If this list is not provided, all source fields are kept in the transformed target file.

For instance with this configuration, the target file will only contains the "name" and "version" fields:

```
field_filters:
- name
- version
```

- **exclude\_fields:** An optional list of field names that should be excluded in the transformed file. If this list is provided, all the fields from the source file that should be excluded in the target file must be listed. Excluding required fields will cause an error. If this list is not provided, all source fields are kept in the transformed target file.

For instance with this configuration, the target file will not contain the "type" and "temp" fields:

```
exclude_fields:
- type
- temp
```

### 2.3.3 Run gen to Generate ABOUT file(s)

When your software inventory is ready, you can save it as a .csv, .json or .xlsx file, and use it as input to run `gen` to generate ABOUT file(s). The official `gen` parameters are defined here: [Reference](#)

Here is an example of a `gen` command:

```
about gen --fetch-license --reference /Users/harrypotter/myLicenseNoticeFiles/
↳ /Users/harrypotter/myAboutFiles/myProject-bom.csv /Users/harrypotter/
↳ myAboutFiles/
```

This `gen` example command does the following:

- Activates the `--fetch-license` option to get license information from ScanCode LicenseDB.
- Activates the `--reference` option to get license text files and notice text files that you have specified in your software inventory to be copied next to the associated .ABOUT files when those are created.
- Specifies the path of the software inventory to control the processing.
- Specifies a target output directory.

Review the generated ABOUT file(s) to determine if it meets your requirements. Here is a simple example of a `linux-redhat-7.2.ABOUT` file that documents the directory `/linux-redhat-7.2/`:

```
about_resource: .
name: Linux RedHat
version: v 7.2
attribute: Y
copyright: Copyright (c) RedHat, Inc.
license_expression: gpl-2.0
licenses:
  - key: gpl-2.0
    name: GPL 2.0
    file: gpl-2.0.LICENSE
    url: https://scancode-licensedb.aboutcode.org/gpl-2.0.LICENSE
    spdx_license_key: GPL-2.0-only
owner: Red Hat
redistribute: Y
```

You can make appropriate changes to your input software inventory and then run `gen` as often as necessary to replace the ABOUT file(s) with the improved version.

## 2.4 Using attrib to Generate a Product Attribution Notice Package

### 2.4.1 Prepare an Attribution Template to Use

You can run `attrib` using the `default_html.template` (or `default_json.template`) provided with the AboutCode Toolkit tools:

[https://github.com/nexB/aboutcode-toolkit/blob/develop/src/attributecode/templates/default\\_html.template](https://github.com/nexB/aboutcode-toolkit/blob/develop/src/attributecode/templates/default_html.template)

If you choose to do that, you will most likely want to edit the generated .html file to provide header information about your own organization and product.

Running `attrib` with the `default_html.template` file is probably your best choice when you are still testing your AboutCode Toolkit process. Once you have a good understanding of the generated output, you can customize the template to

provide the standard text that serve your needs. You can also create alternative versions of the template to use attrib to generate other kinds of documents, such as a License Reference.

## Use jinja2 Features to Customize Your Attribution Template

The attrib tool makes use of the open source python library jinja2 (<http://jinja.pocoo.org/docs/dev/templates/>) in order to extend .html capabilities and transform AboutCode Toolkit input data into the final format of the generated attribution file. `default_html.template` file contains text that complies with jinja2 syntax specifications in order to support grouping, ordering, formatting and presentation of your AboutCode Toolkit data. If your attribution requirements are complex, you may wish to study the jinja2 documentation to modify the `default_html.template` logic or create your own template; alternatively, here are a few relatively simple concepts that relate to the attribution document domain.

The simplest modifications to the `default_html.template` file involve the labels and standard text. For example, here is the default template text for the Table of Contents:

```
<div class="oss-table-of-contents">
  {% for about_object in abouts %}
    <p><a href="#component_{{ loop.index0 }}">{{ about_object.name.value }}
      {% if about_object.version.value %} {{ about_object.version.value }}
      {% endif %}</a></p>
  {% endfor %}
</div>
```

If you would prefer something other than a simple space between the component name and the component version, you can modify it to something like this:

```
<div class="oss-table-of-contents">
  {% for about_object in abouts %}
    <p><a href="#component_{{ loop.index0 }}">{{ about_object.name.value }}
      {% if about_object.version.value %} - Version {{ about_object.
→version.value }}
      {% endif %}</a></p>
  {% endfor %}
</div>
```

The `if about_object.version.value` is checking for a component version, and if one exists it generates output text that is either a space followed by the actual version value, or, as in this customized template, it generates output text as “ - Version “, followed by the actual version value. You will, of course, want to test your output to get exactly the results that you need.

Note that you can actually use attrib to generate an AboutCode Toolkit-sourced document of any kind for varying business purposes, and you may want to change the grouping/ordering of the data for different reporting purposes. (Here we get into somewhat more complex usage of jinja2 features, and you may wish to consult the jinja2 documentation to reach a more comprehensive understanding of the syntax and features.) The default ordering is by component, but In the following example, which is intended to support a “license reference” rather than an attribution document, the customized template modifies the data grouping to use a custom field called “confirmed\_license”:

```
<div class="oss-table-of-contents">
  {% for group in abouts | groupby('confirmed_license') %}
    <p>
      {% for license in group.grouper.value %}
        <a href="#group_{{ loop.index0 }}">{{ license }}
        </a>
      {% endfor %}
    </p>
  {% endfor %}
</div>
```

(continues on next page)

(continued from previous page)

```

    </p>
    {% endfor %}
</div>

```

After the table of contents, this example customized template continues with the license details using the jinja2 for-loop capabilities. Notice that the variable “group.grouper.value” is actually the license name here, and that “License URL” can be any URL that you have chosen to store in your .ABOUT files:

```

{% for group in abouts | groupby('confirmed_license') %}
    {% for confirmed_license in group.grouper.value %}

        <div id="group_{{ loop.index0 }}">
            <h3>{{ confirmed_license }}</h3>
            <p>This product contains the following open source software packages,
            ↳ licensed under the terms of the license: {{confirmed_license}}</p>

            <div class="oss-component" id="component_{{ loop.index0 }}">
                {%for about_object in group.list %}
                    {% if loop.first %}
                        {% if about_object.license_url.value %}
                            {% for lic_url in about_object.license_url.value %}
                                <p>License URL: <a href="{{lic_url}}"
                                ↳ ">{{lic_url }}</a> </p>
                            {% endfor %}
                        {% endif %}
                    {% endif %}
                    <li>
                        {{ about_object.name.value }}{% if about_object.version.value %} -
                        ↳ Version
                        {{ about_object.version.value }}{% endif %}
                    </li>
                    {% if about_object.copyright.value %}<pre>{{about_object.copyright.
                    ↳ value}}</pre>{% endif %}
                    {% if loop.last %}
                        <pre>
                            {% for lic_key in about_object.license_file.value %}
                                {{about_object.license_file.value[lic_key]}}
                            {% endfor %}
                        </pre>
                    {% endif %}
                {% endfor %}
            </div>
            <hr>
        </div>
    {% endfor %}
</hr>

```

In summary, you can start with simple, cosmetic customizations to the default\_html.template, and gradually introduce a more complex structure to the attrib output to meet varying business requirements.

## 2.4.2 Run attrib to Generate a Product Attribution Notice Package

You can then run the `attrib` to generate your product attribution notice package from the generated ABOUT file(s) or from an inventory (.csv/.json/.xlsx). The official `attrib` parameters are defined here: [Reference](#)

Here is an example of a `attrib` command:

```
about attrib --template /Users/harrypotter/myAboutFiles/my_attribution_template_v1.html /Users/harrypotter/myAboutFiles/ /Users/harrypotter/myAboutFiles /myProject-attribution-document.html
```

Note that this example `attrib` command does the following:

- Activates the `--template` option to specify a custom output template.
- Specifies the path of the ABOUT file(s) that use to generate the output attribution.
- Specifies the full path (include file name) of the output document to be generated.

Another example:

```
about attrib /Users/harrypotter/inventory.xlsx /Users/harrypotter/attribution.html --reference /Users/harrypotter/licenses/
```

The above command does the following:

- Use the `inventory.xlsx` as the input
- Specifies the location of the generated output document
- Specifies the `licesen_file` or `notice_file` location that can be found in the `--reference` option

A successful execution of `attrib` will create a .html (or .json depends on the template) file that is ready to use to meet your attribution requirements.

Please refer to the `attrib` section in [Reference](#) for more information.

## 2.5 Using inventory to Generate a Software Inventory

### 2.5.1 Generate a Software Inventory of Your Codebase from ABOUT file(s)

One of the major features of the ABOUT File specification is that the .ABOUT files are very simple text files that can be created, viewed and edited using any standard text editor. Your software development and maintenance processes may require or encourage your software developers to maintain .ABOUT files and/or associated text files manually. For example, when a developer addresses a software licensing issue with a component, it is appropriate to adjust the associated ABOUT file(s) manually.

If your organization adopts the practice of manually creating and maintaining ABOUT file(s), you can easily re-create your software inventory from your codebase using inventory. The official inventory parameters are defined here: [Reference](#)

A successful execution of `inventory` will create a complete software inventory in .csv, .json or .xlsx format based on defined format.



## ABOUT FILE SPECIFICATION V3.3.2

### 3.1 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.

### 3.2 Getting Started

A simple and valid ABOUT file named `httpd-2.4.3.tar.gz.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
    name: Apache 2.0
    file: apache-2.0.LICENSE
    url: https://scancode-licensedb.aboutcode.org/apache-2.0.LICENSE
    spdx_license_key: Apache-2.0
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-2.4.3.tar.gz.ABOUT`” that documents it.
- The name of this component is “Apache HTTP Server” with version “2.4.3”.
- The homepage 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>
- This component is licensed under “apache-2.0”
- The licenses section contains the information of this “apache-2.0” license.
- 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.

## 3.3 Specification

An ABOUT file is an YAML formatted text file. The key for the licenses field and the license\_expression are ScanCode license key.

### 3.3.1 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 any characters and digits with the following exception and condition:

- the following symbols are not accepted: ", #, &, ', \*, \, :, ;, <, >, =, ?, /, ^, `, |
- 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.

### 3.3.2 Lines of text

An ABOUT file contains lines of 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.

### 3.3.3 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. No spaces is allowed in the field name. 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.

### 3.3.4 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 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.
```

### 3.3.5 Fields are mandatory, optional or custom extension

A field can be mandatory, optional or custom extension. Tools must report an error for missing mandatory fields.

### 3.3.6 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 URL validation, path resolution and verification, and so forth. Tools should report a warning for present fields that do not have any value.

### 3.3.7 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/XLSX input will issue an error when a field name occurs more than once in the input file.

### 3.3.8 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. If the field is referencing a license file, a “file” field within the “licenses” group can be used. This path must be a POSIX path relative to the path of the ABOUT file. The file content must be UTF-8-encoded text.

For example, this example shows the license file for the component is named “linux.COPYING” and the notice file is “NOTICE”:

```
license_file: linux.COPYING
notice_file: NOTICE
```

Alternatively, it can also write as the follow:

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

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
```

In addition, there may be cases that a license can have 2 or more referenced license files. If this is the case, a comma ‘,’ is used to identify multiple files For instance:

```
license_expression: gpl-2.0-plus
licenses:
  - key: gpl-2.0-plus
    file: COPYING, COPYING.LESSER
```

### 3.3.9 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
```

### 3.3.10 Flag fields

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

### 3.3.11 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. There is also a `ignored_resources` field which can be used to ignore a set of subpaths inside the directory which is being documented in the ABOUT file.

A tool processing an ABOUT file must report an error if the `about_resource` 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 a whole sub-directory, with some sub-paths under the directory ignored:

```
about_resource: linux-kernel-2.6.23
ignored_resources: linux-kernel-2.6.23/Documentation
```

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

```
about_resource: .
```

### 3.3.12 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.

### 3.3.13 Optional Information fields

- `ignored_resources`: A list of paths under the `about_resource` path, which are not documented in the ABOUT file, and the information in the ABOUT file does not apply to these subpaths.
- `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.

- `package_url`: Package URL for the package.
- `notes`: Notes and comments about the component.

### 3.3.14 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.

### 3.3.15 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 ScanCode 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 (No special characters are allowed).
- `license_name`: The ScanCode license short name for the license (No special characters are allowed).
- `license_key`: The ScanCode license key(s) for the component (No special characters are allowed).
- `spdx_license_key`: The ScanCode LicenseDB `spdx_license_key` defined for the license at <https://scancode-licensedb.aboutcode.org/index.html>
- `spdx_license_expression`: The license expression that use `spdx_license_key`

### Notes

The `license_*` fields in the generated `.ABOUT` files are grouped under the “licenses” fields. For instance,

```
licenses:
-   key: apache-2.0
    name: Apache 2.0
    file: apache-2.0.LICENSE
    url: https://scancode-licensedb.aboutcode.org/apache-2.0.LICENSE
    spdx_license_key: Apache-2.0
```

However, if user create `.ABOUT` file manually, it can also used the individual field name.

```
license_key: apache-2.0
license_name: Apache 2.0
license_file: apache-2.0.LICENSE
license_url: https://scancode-licensedb.aboutcode.org/apache-2.0.LICENSE
spdx_license_key: Apache-2.0
```

These grouping is only used in the generated .ABOUT files. The output from **gen** will use the individual field name.

### 3.3.16 Optional Boolean flag fields

- **redistribute**: Set this flag to yes if the component license requires source code redistribution. 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.

### 3.3.17 Optional Boolean and Character fields

- **attribute**: This field can be either in boolean value: ('yes', 'y', 'true', 'x', 'no', 'n', 'false') or a character value field with no more than 2 characters. Defaults to no when absent.

### 3.3.18 Optional Extension fields

You can create extension fields by prefixing them with a short prefix to distinguish these from the standard fields (but this is not necessary).

### 3.3.19 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
```

### 3.3.20 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.
- `checksum_sha256`: SHA256 for the file documented by this ABOUT file in the “`about_resource`” field.

Some examples:

```
checksum_md5: f30b9c173b1f19cf42ffa44f78e4b96c
```



## REFERENCE

## 4.1 about

### 4.1.1 Syntax

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

### 4.1.2 Options

<code>--version</code>	Show the version and exit.
<code>-h, --help</code>	Show this message and exit.

### 4.1.3 Commands

<code>attrib</code>	Generate an attribution document from JSON/CSV/XLSX/.ABOUT files.
<code>check</code>	Validate that the format of .ABOUT files is correct and report errors and warnings.
<code>collect-redist-src</code>	Collect redistributable sources.
<code>gen</code>	Generate .ABOUT files from an inventory as CSV/JSON/XLSX.
<code>gen-license</code>	Fetch and save all the licenses in the <code>license_expression</code> field to a directory.
<code>inventory</code>	Collect the inventory of .ABOUT files to a CSV/JSON/XLSX file.
<code>transform</code>	Transform a CSV/JSON/XLSX by applying renamings, filters and checks.

## 4.2 attrib

### 4.2.1 Syntax

```
about attrib [OPTIONS] LOCATION OUTPUT
```

INPUT: Path to a file (.ABOUT/.csv/.json/.xlsx), directory or .zip archive, containing .ABOUT files.

OUTPUT: Path where to write the attribution document.

### 4.2.2 Options

```
--api_url URL           URL to DejaCode License Library.
--api_key KEY           API Key for the  DejaCode License Library
--min-license-score INTEGER Attribute components that have license score
                        higher than or equal to the defined --min-
                        license-score.
--scancode              Indicate the input JSON file is from
                        scancode_toolkit.
--reference DIR         Path to a directory with reference files where
                        "license_file" and/or "notice_file" located.
--template FILE         Path to an optional custom attribution template
                        to generate the attribution document. If not
                        provided the default built-in template is used.
--vartext <key>=<value> Add variable text as key=value for use in a
                        custom attribution template.
--worksheet name        The worksheet name from the INPUT. (Default:
                        the "active" worksheet)
-q, --quiet             Do not print error or warning messages.
--verbose              Show all error and warning messages.
-h, --help             Show this message and exit.
```

### 4.2.3 Purpose

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

Assume the following:

```
'/home/about_files/' contains all the ABOUT files [INPUT]
'/home/project/inventory.csv' is a BOM inventory [INPUT]
'/home/project/scancode-detection.json' is a detection output from scancode-
→ toolkit[INPUT]
'/home/project/licenses/' contains all the license/notice file references
'/home/attribution/attribution.html' is the user's output path [OUTPUT]
```

```
$ about attrib /home/about_files/ /home/attribution/attribution.html
or
$ about attrib /home/project/inventory.csv /home/attribution/attribution.html -
```

(continues on next page)

(continued from previous page)

```

↪--reference /home/project/licenses/
or
$ about attrib --scancode /home/project/scancode-detection.json /home/
↪attribution/attribution.html

```

## Details

```

--api_url URL --api_key

    This option let user to define where to get the license information such as
    from DJE. If these options are not set, the tool will get the license
    information from ScanCode LicenseDB by default

$ about attrib --api_url <URL> --api_key <KEY> INPUT OUTPUT

--min-license-score

    This option is a filter to collect license information where the license
    ↪score
    in the scancode toolkit detection is greater than or equal to the defined
    --min-license-score. This option is specifically design for scancode's
    ↪input
    and therefore --scancode is required

$ about attrib --scancode --min-license-score 85 /home/project/scancode-
↪detection.json OUTPUT

--reference

    This option is to define the reference directory where the 'license_file'
    or 'notice_file' are stored

$ about attrib --reference /home/project/licenses/ /home/project/inventory.csv
↪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 INPUT 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

```

(continues on next page)

(continued from previous page)

```

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

--worksheet

    This option identify the worksheet name from the XLSX input to work with.
    If no worksheet is defined, the "active" worksheet will be used

$ about attrib --worksheet BOM /home/project/audit.xlsx OUTPUT

--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`
- `licenses_list`: a license object list contains all the licenses found in about objects. It contains the following attribute: `key`, `name`, `filename`, `url`, `text`

## 4.3 check

### 4.3.1 Syntax

```
about check [OPTIONS] LOCATION
```

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

### 4.3.2 Options

<code>--license</code>	Validate the <code>license_expression</code> value in the input.
<code>--djv api_url api_key</code>	Validate <code>license_expression</code> from a DejaCode License Library API URL using the API KEY.
<code>--log FILE</code>	Path to a file to save the error messages if any.
<code>--verbose</code>	Show all error and warning messages.
<code>-h, --help</code>	Show this message and exit.

### 4.3.3 Purpose

Validating ABOUT files at LOCATION.

#### Details

```
--license
    Validate the license_expression value in the input.

    If this option is not flagged, only the basic syntax is checked.
    No validation of the license_expression value.

$ about check --license /home/project/about_files/

---djv

    Validate license_expression from a DejaCode License.

    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 for this feature)

$ about check --license --djv 'api_url' 'api_key' /home/project/about_files/

--log

    This option save the error log to the defined location

$ about check --log /home/project/error.log /home/project/about_files/

--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/
```

### 4.3.4 Special Notes

If no `-djc` option is set, the tool will default to check `license_expression` from ScanCode LicenseDB.

## 4.4 collect\_redist\_src

### 4.4.1 Syntax

```
about collect_redist_src [OPTIONS] LOCATION OUTPUT
```

LOCATION: Path to a directory containing sources that need to be copied (and containing ABOUT files if ``inventory`` is not provided)

OUTPUT: Path to a directory or a zip file where sources will be copied to.

### 4.4.2 Options

```
--from-inventory FILE  Path to an inventory CSV/JSON/XLSX file as the base
                        list for files/directories that need to be copied
                        which have the 'redistribute' flagged.
--with-structures      Copy sources with directory structure.
--zip                  Zip the copied sources to the output location.
-q, --quiet            Do not print error or warning messages.
--verbose              Show all error and warning messages.
-h, --help             Show this message and exit.
```

### 4.4.3 Purpose

Collect sources that have `'redistribute'` flagged as `'True'` in `.ABOUT` files or inventory to the output location.

#### Details

```
--from-inventory
```

Provide an inventory CSV/JSON file with the `'redistribute'` field filled as the indication of which files/sources need to be copied.

```
$ about collect_redist_src --from-inventory 'path to the inventory' LOCATION_
↪ OUTPUT
```

```
--with-structures
```

Copy the file(s) along with its parent directories

For instance, assuming we want to copy the following file:  
`/project/work/hello/foo.c`

(continues on next page)

(continued from previous page)

```

    OUTPUT: /output/

$ about collect_redist_src --with-structure /project/ /output/

    OUTPUT: /output/work/hello/foo.c

$ about collect_redist_src /project/ /output/

    OUTPUT: /output/foo.c

--zip

    Zip the copied sources to the output location

$ about collect_redist_src --zip /project/ /output/output.zip

--verbose

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

```

## 4.5 gen

### 4.5.1 Syntax

```

about gen [OPTIONS] LOCATION OUTPUT

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

```

### 4.5.2 Options

--android	Generate MODULE_LICENSE_XXX (XXX will be replaced by license key) and NOTICE as the same design as from Android.
--fetch-license	Fetch license data and text files from the ScanCode LicenseDB.
--fetch-license-djc api_url api_key	Fetch license data and text files from a DejaCode License Library API URL using the API KEY.
--reference DIR	Path to a directory with reference license data and text files.
--worksheet name	The worksheet name from the INPUT. (Default: the "active" worksheet)
-q, --quiet	Do not print error or warning messages.
--verbose	Show all error and warning messages.
-h, --help	Show this message and exit.

### 4.5.3 Purpose

Given a CSV/JSON/XLSX inventory, generate ABOUT files in the output location.

#### Details

```
--android

    Create an empty file named `MODULE_LICENSE_XXX` where `XXX` is the license
    key and create a NOTICE file which these two files follow the design from
    Android Open Source Project.

    The input **must** have the license key information as this is needed to
    create the empty MODULE_LICENSE_XXX

$ about gen --android LOCATION OUTPUT

--fetch-license

    Fetch licenses text and create <license>.LICENSE side-by-side
    with the generated .ABOUT file using the data fetched from the the
    ↪ ScanCode LicenseDB.

    The input needs to have the 'license_expression' field.

$ about gen --fetch-license LOCATION OUTPUT

--fetch-license-djc

    Fetch licenses text from a DejaCode API, and create <license>.LICENSE side-
    ↪ by-side
    with the generated .ABOUT file using the data fetched from the 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 for this feature)

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

--reference

    Copy the reference files such as 'license_files' and 'notice_files' to the
    generated location from the specified directory.

    For instance,
    the specified directory, /home/licenses_notices/, contains all the
    ↪ licenses and notices:
    /home/licenses_notices/apache2.LICENSE
```

(continues on next page)



(continued from previous page)

```

/home/licenses_notices/jquery.js.NOTICE

$ about gen --reference /home/licenses_notices/ LOCATION OUTPUT

--worksheet

    This option identify the worksheet name from the XLSX input to work with.
    If no worksheet is defined, the "active" worksheet will be used

$ about gen --worksheet BOM LOCATION OUTPUT

--verbose

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

```

#### 4.5.4 Special Notes

If the input contains values for `license_file`, the tool will attempt to associate the `license_file` with the corresponding `license_key`.

sample.csv

about_resource	name	license_expression	license_file
/project/test.c	test.c   mit AND custom		custom.txt

If the user does not utilize the **-fetch-license** option, the input will contain two license keys and one license file. In this scenario, the tool cannot determine which license key the license file is referencing. As a result, the `license_file` will be saved separately.

i.e.

```

about_resource: test.c
name: test.c
license_expression: mit AND custom
licenses:
  - key: mit
    name: mit
  - key: custom
    name: custom
  - file: custom.txt

```

On the other hand, if the user generates ABOUT files using the **-fetch-license** option, the MIT license will be retrieved. This will result in having one license key and one license file. In such cases, the tool will consider it a successful match.

i.e.

```

about_resource: test.c
name: test.c
license_expression: mit AND custom
licenses:

```

(continues on next page)

(continued from previous page)

```
- key: mit
  name: MIT License
  file: mit.LICENSE
  url: https://scancode-licensedb.aboutcode.org/mit.LICENSE
  spdx_license_key: MIT
- key: custom
  name: custom
  file: custom.txt
```

## 4.6 gen\_license

### 4.6.1 Syntax

```
about gen_license [OPTIONS] LOCATION OUTPUT
```

LOCATION: Path to a JSON/CSV/XLSX/.ABOUT file(s)

OUTPUT: Path to a directory where license files are saved.

### 4.6.2 Options

```
--djv api_url api_key  Fetch licenses from a DejaCode License Library.
--scancode              Indicate the input JSON file is from
                        scancode_toolkit.
--worksheet name        The worksheet name from the INPUT. (Default: the
                        "active" worksheet)
--verbose               Show all error and warning messages.
-h, --help              Show this message and exit.
```

### 4.6.3 Purpose

Fetch licenses (Default: ScanCode LicenseDB) in the license\_expression field and save to the output location.

#### Details

```
--djv

Fetch licenses text from a DejaCode API, and create <license>.LICENSE to
the
OUTPUT Location using the data fetched from the 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.
```

(continues on next page)

(continued from previous page)

```
(Please contact nexB to get the api_* value for this feature)

$ about gen_license --djv 'api_url' 'api_key' LOCATION OUTPUT

--scancode

    Indicates the JSON input is from scancode toolkit license detection

$ about gen_license --scancode /home/project/scancode-license-detection.json
↪ OUTPUT

--worksheet

    This option identify the worksheet name from the XLSX input to work with.
    If no worksheet is defined, the "active" worksheet will be used

$ about gen_license --worksheet BOM /home/project/bom-v0.10.xlsx OUTPUT

--verbose

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

## 4.6.4 Special Notes

If no `-djv` option is set, the tool will default to fetch licenses from ScanCode LicenseDB.

## 4.7 inventory

### 4.7.1 Syntax

```
about inventory [OPTIONS] LOCATION OUTPUT

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

### 4.7.2 Options

<code>-f, --format [json csv excel]</code>	Set OUTPUT file format. [default: csv]
<code>-q, --quiet</code>	Do not print any error/warning.
<code>--verbose</code>	Show all the errors and warning.
<code>-h, --help</code>	Show this message and exit.

### 4.7.3 Purpose

Create a JSON/CSV/XLSX inventory of components from ABOUT files.

#### Details

```
-f, --format [json|csv|excel]
```

Set OUTPUT file format. [default: csv]

```
$ about inventory -f json LOCATION OUTPUT
```

```
--verbose
```

This option tells the tool to show all errors found.

The default behavior will only show 'CRITICAL', 'ERROR', and 'WARNING'

### 4.7.4 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_name	license_file
test.tar.xz	test	apache-2.0 mit	Apache 2.0 MIT License	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
    file: apache-2.0.LICENSE
  - key: mit
    name: MIT License
    file: mit.LICENSE
```

## Multiple license\_file support

To support multiple license file for a license, the correct format is to separate by comma

about_resource	name	license_key	license_name	license_file
test.tar.xz	test	gpl-2.0 mit	GPL 2.0 MIT License	COPYING, COPYINGv2 mit.LICENSE

```
about_resource: test.tar.xz
name: test
licenses:
  - key: gpl-2.0
    name: gpl-2.0
    file: COPYING, COPYING.v2
  - key: mit
    name: mit
    file: mit.LICENSE
```

Note that if license\_name is not provided, the license key will be used as the license name.

## 4.8 transform

### 4.8.1 Syntax

```
about transform [OPTIONS] LOCATION OUTPUT

LOCATION: Path to a CSV/JSON/XLSX file.
OUTPUT: Path to CSV/JSON/XLSX inventory file to create.
```

### 4.8.2 Options

```
-c, --configuration FILE  Path to an optional YAML configuration file. See
                           --help-format for format help.
--worksheet name          The worksheet name from the INPUT. (Default: the
                           "active" worksheet)
--help-format             Show configuration file format help and exit.
-q, --quiet               Do not print error or warning messages.
--verbose                 Show all error and warning messages.
-h, --help                Show this message and exit.
```

### 4.8.3 Purpose

Transform the CSV/JSON/XLSX file at LOCATION by applying renamings, filters and checks and then write a new CSV/JSON/Excel to OUTPUT.

#### Details

```
-c, --configuration

    Path to an optional YAML configuration file. See--help-format for format.
↪help.

$ about transform -c 'path to the YAML configuration file' LOCATION OUTPUT

--worksheet

    This option identify the worksheet name from the XLSX input to work with.
    If no worksheet is defined, the "active" worksheet will be used

$ about transform -c 'path to the YAML configuration file' --worksheet BOM /
↪project/bom-v.20.xlsx OUTPUT

--help-format

    Show configuration file format help and exit.
    This option will print out examples of the the YAML configuration file.

    Keys configuration are: `field_renamings`, `required_fields` and `field_
↪filters`

$ about transform --help-format

--verbose

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

### 4.8.4 --help-format

```
A transform configuration file is used to describe which transformations and
validations to apply to a source CSV file. This is a simple text file using
↪YAML
format, using the same format as an .ABOUT file.

The attributes that can be set in a configuration file are:

* field_renamings:
An optional map of source CSV or JSON field name to target CSV/JSON new field.
↪name that
is used to rename CSV fields.
```

(continues on next page)

(continued from previous page)

For instance with this configuration the fields "Directory/Location" will be renamed to "about\_resource" and "foo" to "bar":

```
field_renamings:
  about_resource : 'Directory/Location'
  bar : foo
```

The renaming is always applied first before other transforms and checks. All other field names referenced below are these that exist AFTER the renamings have been applied to the existing field names.

#### \* required\_fields:

An optional list of required field names that must have a value, beyond the standard fields names. If a source CSV/JSON does not have such a field or a row is missing a value for a required field, an error is reported.

For instance with this configuration an error will be reported if the fields "name" and "version" are missing or if any row does not have a value set for these fields:

```
required_fields:
  - name
  - version
```

#### \* field\_filters:

An optional list of field names that should be kept in the transformed CSV/JSON. If this list is provided, all the fields from the source CSV/JSON that should be kept in the target CSV/JSON must be listed regardless of either standard or required fields. If this list is not provided, all source CSV/JSON fields are kept in the transformed target CSV/JSON.

For instance with this configuration the target CSV/JSON will only contains the "name" and "version" fields and no other field:

```
field_filters:
  - name
  - version
```

#### \* exclude\_fields:

An optional list of field names that should be excluded in the transformed CSV/JSON. If this list is provided, all the fields from the source CSV/JSON that should be excluded in the target CSV/JSON must be listed. Excluding standard or required fields will cause an error. If this list is not provided, all source CSV/JSON fields are kept in the transformed target CSV/JSON.

(continues on next page)

(continued from previous page)

```

For instance with this configuration the target CSV/JSON will not contain the
↪ "type"
and "temp" fields:
  exclude_fields:
    - type
    - temp

```

## 4.8.5 Example

### fields renaming

#### conf.txt

```

field_renamings:
  about_resource : 'Directory / Filename'
  name : Component
  version: 'Confirmed Version'
  license_expression: 'Confirmed License Expression'

```

#### input.csv

Directory / Filename	Component	Confirmed Version	Confirmed License Expression
/project/sample/	sample	v 1.2.3	apache-2.0

### Command

```
about transform -c conf.txt input.csv output.csv
```

The result output will look like the following:

#### output.csv

about_resource	name	version	license_expression
/project/sample/	sample	v 1.2.3	apache-2.0



### 4.8.6 Special Notes

When using the `field_filters` configuration, all the standard required columns (`about_resource` and `name`) and the user defined `required_fields` need to be included.

## 4.9 Notes

The AboutCode Toolkit version 10.0.0 will work with input from Scancode Toolkit version 32.0.0 or later. If you are using an earlier version of Scancode Toolkit, specifically version 31 or older, it will only be compatible with prior versions of AboutCode Toolkit.

### 4.9.1 Configure proxy

The *requests* library is used since AboutCode Toolkit version 10.1.0. To do the http request, users can set the standard environment variables **http\_proxy**, **https\_proxy**, **no\_proxy**, **all\_proxy** with the export statement

i.e.

```
$ export HTTP_PROXY="http://10.10.1.10:3128"
$ export HTTPS_PROXY="http://10.10.1.10:1080"
$ export ALL_PROXY="socks5://10.10.1.10:3434"
```

See <https://requests.readthedocs.io/en/latest/user/advanced/#proxies> for references

## TYPE OF ERRORS

We have 6 type of errors as describe below:

### 5.1 NOTSET

#### 5.1.1 Trigger:

- None

#### Details

We do not have event to trigger this error.

### 5.2 DEBUG

#### 5.2.1 Trigger:

- None

#### Details

We do not have event to trigger this error.

### 5.3 INFO

#### 5.3.1 Trigger:

- *about\_resource* not found
- Custom fields detected
- Empty field value

## 5.4 WARNING

### 5.4.1 Trigger:

- Duplicated value being ignored
- Invalid Package URL from input
- Invalid URL from input

## 5.5 ERROR

### 5.5.1 Trigger:

- Invalid license
- Invalid API call
- Invalid character
- Invalid input
- Duplicated field name
- Incorrect input format
- Failure to write ABOUT file
- Network problem

## 5.6 CRITICAL

### 5.6.1 Trigger:

- Invalid template
- File field not found
- Duplicated *about\_resource*
- Not supported field format
- Essential or required field not found
- Internal error
- Empty ABOUT file
- Invalid ABOUT file

---

**Note:** If *-verbose* is set, all the detected errors will be reported. Otherwise, only “CRITICAL”, “ERROR” and “WARNING” will be reported.

---