



Automated License Compliance

1	Contents	
2	FOSSology	3
3	CI Pipeline integration	3
4	Binary to source file mapping	4
5	CI Pipeline integration	5
6	Binary Licensing Reporting	6
7	CI Pipeline integration	6
8	A Linux system such as those assembled by Apertis contain components licensed	
9	under many different licenses. These various licenses impose different conditions	
10	and it is important to understand to a good degree of fidelity the terms under	
11	which each component is provided. We are proposing to implement an auto-	
12	mated process to generate software Bills Of Materials (BOMs) which detail	
13	both the components used in Apertis and the licensing that applies to them.	
14	Licensing isn't static, nor is it always as simple as all the components from a	
15	given source package deriving the same license. Packages have been known to	
16	change licenses and/or provide various existing or new components under dif-	
17	ferent terms. Either now or at some point in the future, the licenses of some of	
18	the components in Apertis may start to be provided under terms that Apertis	
19	may wish to avoid ¹ . For example, by default Apertis is careful not to include	
20	components to be used in the target system that are licensed under the GPL	
21	version 3, the licensing terms wouldn't be acceptable in Apertis' target markets.	
22	In order to take advantage of new functionality and support being developed in	
23	the software community, Apertis needs to incorporate newer versions of exist-	
24	ing software packages and replace some with alternatives when better or more	
25	suitable components are created. To ensure that the licensing conditions remain	
26	favorable for the use cases targeted by Apertis, it is important to continually	
27	validate that the licensing terms under which these components are provided.	
28	These licensing terms should be documented in a way that is accessible to Aper-	
29	tis' users.	
30	Debian packages by default track licensing on a per source package level. The	
31	suitability of a package is decided at that level before it is included in Debian,	
32	which meets the projects licensing goals ² . Apertis will continue to evaluate	
33	licensing before the inclusion of source packages in the distribution, but also	
34	wishes to take a more nuanced approach, tracking licensing for each file in each	
35	of it's binary packages. By tracking licensing to this degree we can look to	
36	exclude components with unsatisfactory licensing from the packages intended	
37	for distributed target systems, whilst still packaging them separately so they	
38	may be utilized during development. A good example of this situation is the	
39	gcc source package and the libgcc1 binary package produced by it. Unlike the	
40	other artifacts produced by the GCC source package, the libgcc1 binary package	

¹<https://jwd.pages.apertis.org/apertis-website/policies/license-expectations/>

²https://www.debian.org/social_contract.html#guidelines

41 is not licensed under the stock GPLv3 license, a [run time exception](#)³ is provided
42 and it is thus fine to ship it on target devices. The level of tracking we are
43 providing will detect such situations and will offer a straight forward way to
44 resolve them, maintaining compliance with the licensing requirements.

45 To achieve this 2 main steps need to be taken:

- 46 • Record the licensing of the project source code, per file
- 47 • Determine the mapping between source code files and the binary/data
48 files in each binary package

49 These steps have been integrated into our CI pipelines to provide early detection
50 of any change to the licensing status of each package. Extending our CI pipelines
51 also enables developers to learn about new issues and to solve them during the
52 merge request development flow.

53 FOSSology

54 FOSSology is an Open Source server based tool which provides a web front-end
55 that is able to scan through source code (and to a degree binaries) provided to
56 it, finding license statements and texts. To achieve this FOSSology employs a
57 number of different scanning techniques to identify potential licenses, including
58 using matching to known license texts and keywords. The scanning process errs
59 on the side of caution, generating false positives over missing potential licens-
60 ing information, as a result it will be necessary to “clear” the licenses that are
61 found, deciding whether the matches are valid or not. The scanning and clear
62 process during the first time is more time consuming and requires special atten-
63 tion, however, subsequent runs should be much faster as FOSSology is able to
64 use previous decisions to find the license information. Once completed, FOSSol-
65 ogy records the licensing decisions and can apply this information to updated
66 scans of the source. It is anticipated that, after an initial round of verification,
67 FOSSology will only require additional clearing of license information should
68 the scan detect new sources of potential licensing information in an updated
69 projects source or when new packages are added to Apertis. It is possible to
70 export and import reports which contain the licensing decisions that have pre-
71 viously been made, if a trusted source of reports can be found then these could
72 also be imported, potentially reducing the work required.

73 FOSSology is backed by the Linux Foundation, it appears to have an active user
74 and developer base and a significant history and it is the de-facto Open Source
75 Software solution for license compliance. As such, it is felt that this tool is likely
76 to be maintained for the foreseeable future.

77 As this tool provides a web bases UI, it presents an additional advantage, as
78 it makes it easier for non-technical users, such as auditors or lawyers, to access
79 and manage the reports, allowing a smooth integration in an audit process.

³<https://jwd.pages.apertis.org/apertis-website/policies/license-exceptions/#gcc8>

80 For all the reasons mentioned above we understand this is the best choice for
81 integration into the Apertis workflow.

82 CI Pipeline integration

83 In order to avoid manual tasks the license detection should be integrated into
84 the CI process. FOSSology provides a [REST API](#)⁴ to enable such integration.

85 FOSSology is able to consume branches of git repositories, thus allowing scan-
86 ning of the given source code straight from GitLab. This process should be
87 triggered after updating a package from external sources, as in this cases a
88 license change can be introduced. A report will be generated and retrieved, us-
89 ing the REST API, which describes (among other things) the licensing status of
90 each file. The report can be generated in a number of formats, including various
91 SPDX flavors that are easily machine parsable, using [DEP5](#)⁵ as the preferred
92 option. It is suggested that each component should require a determination of
93 the licensing to have been made for every file in the project. Due to the large
94 volume of licensing matches that will result from the initial licensing scan, we
95 recommend that the absence of license information initially generates a warn-
96 ing. In some cases, to achieve the fine grained licensing information desired, the
97 licensing of some files may need to be clarified with the components author(s).
98 Once an initial pass of all Apertis components had been made we would expect
99 missing license information to result in an error, as such errors would be as a
100 result of new matches being found, which would need to be resolved in FOSSol-
101 ogy before CI would complete without an error. The generated report should
102 be saved in the Debian metadata archive so that it is available for the following
103 processing.

104 The adoption of FOSSology will be gradual and in parallel with the current
105 license scanning process in order to compare the results and improve the work-
106 flow. Once the process is fully reviewed and tested with all the packages in the
107 target repository FOSSology will be the default scanner.

108 Binary to source file mapping

109 Binaries are built from many different source files, but the exact list of them
110 depends on build options. For this reason a reliable mechanism needs to be put
111 in place to extract this list after the build process in order to determine the
112 license information.

113 Compilers store information in the binaries it outputs, that can be used by a
114 debugger to pause execution of a process at a point corresponding to a selected
115 line of source code. This information provides a mapping between the lines of
116 source code and the compiled machine code operations. Executable binaries

⁴<https://www.fossology.org/get-started/basic-rest-api-calls/>

⁵<https://dep-team.pages.debian.net/deps/dep5/>

117 in Linux are generally stored in the [Executable and Linkable Format](#)⁶ (ELF),
118 the associated [DWARF](#)⁷ debugging data format is generally used to store this
119 debugging information inside the ELF in specific “debug” sections.

120 The tool `dwarf2sources` parses this information and extracts the name of the
121 source files that were used to generate each binary, generating a `json` file that can
122 easily be parsed later. Combining this with the licensing information provided
123 in the licensing report, a mapping can be made between each binary and it’s
124 associated licenses.

125 CI Pipeline integration

126 Apertis uses the Open Build Service (OBS) platform to build the binary pack-
127 ages in a controlled manner across several architectures and releases. OBS uti-
128 lizes `dpkg-buildpackage` behind the scenes to build each package. This utility has
129 access to the source licensing report as it is contained in the Debian metadata
130 archive. As well as the source licensing, the Debian metadata archive contains
131 configuration to help `dpkg-buildpackage` determine how to build the source. This
132 is typically done with the help of [debhelper](#)⁸, which provides helpers that sim-
133 plify this process.

134 Apertis extended `debhelper` by including a new command `dh_dwarf2sources` to
135 perform the source file name extraction using `dwarf2sources` as described above.
136 Typically the binaries are striped (using a `debhelper` command called `dh_strip`)
137 prior to packaging, removing the debug symbols from the binary and reducing
138 its size. For this reason `dh_dwarf2sources` is placed before this step in the `dh`
139 sequence. Whilst the debug symbols are kept, packaged separately in the `dbg` package,
140 it’s easier to perform the mapping before this is done. The result is
141 stored in the binary package under `/usr/share/doc/<package>/`.

142 Following this same idea, Apertis also extends `debhelper` command
143 `dh_installdocs` to install the license report generated by FOSSolgy in the
144 binary under `/usr/share/doc/<package>/copyright_report`.

145 Despite that this solution should work for most packages, some of them might
146 need special handing as may override default rules. These special cases will be
147 covered with further improvements.

148 There may be packages in Apertis that do not make use of `debhelper`, these
149 packages need special handling to ensure that the required steps are completed.

150 As these reports are provided by each binary package, the reports from installed
151 packages can be accessed at image build time and amalgamated into an image
152 wide report at that point should it be required. As a binary can be built from
153 multiple sources, each with differing licenses, it is necessary for the report to
154 detail each file that is used to create each binary and the licensing under which

⁶https://en.wikipedia.org/wiki/Executable_and_Linkable_Format

⁷<https://en.wikipedia.org/wiki/DWARF>

⁸<https://manpages.debian.org/jessie/debhelper/debhelper.7.en.html>

155 it is provided. In some circumstances dual licensed source code may allow for
156 a binary to be effectively licensed under the terms of a single license, that is
157 the user has the option to pick a license that results in the whole binary being
158 able to be provided under the terms of a single license. Where dual licensed
159 source code isn't used, the terms of all applicable licenses should be declared.
160 The terms of the various licenses may be considered [compatible](#)⁹, allowing the
161 binary to effectively be managed under the terms of the more restrictive license.
162 For example, a binary derived from source code licensed with the GPLv2 license
163 and other source code licensed with the MIT license, the terms of both apply to
164 the binary, though as the terms of the MIT license will be met if the binary is
165 used in accordance with the terms of the GPLv2, then handling the binary as
166 though it was licensed under the GPLv2 will ensure the terms of both are met.
167 Not all possible combinations of licenses work out this way and thus why it is
168 important to ensure that licensing is properly tracked.

169 Binary Licensing Reporting

170 The approach each project using Apertis takes with regards to the reporting of
171 licensing information should be driven by how this information is to be utilized,
172 i.e. some projects may wish to parse the license information and present it in a
173 single BOM file in HTML, XML or human readable text.

174 For the images provided by the Apertis project, the script `generate_bom.py` com-
175 bines the reports saved in `/usr/share/doc/<package>/`, which consists in a `json`
176 per package and a `DEP5` file per source package into a single `json` file which is
177 provided with the image. This file can be generated with different levels of
178 verbosity allowing to list licenses per image, package, binary or source file.

179 This same scripts also issues a warning in case a problematic license is found.

180 CI Pipeline integration

181 Apertis utilizes [Debos](#)¹⁰ in its image generation pipeline, which provides a very
182 versatile way of customizing them. During the final stage of the image creation,
183 the script `generate_bom.py` is used to build the BOM file with the license informa-
184 tion of the image and export it as an additional artifact. Finally as both `minimal`
185 and `targetimages` should not shipped extra data, the contents of `/usr/share/doc/`
186 are dropped from the image.

⁹https://en.wikipedia.org/wiki/License_compatibility

¹⁰<https://github.com/go-debos/debos>