NGI Zero PET and Discovery Legal To-Dos

Free Software Licensing
Frequently Asked Questions and
Background Information

Published by the Free Software Foundation Europe (FSFE)
Berlin, September 2019
EU Transparency register ID: 33882407107-76
www.fsfe.org

Responsible according to European press law:
Matthias Kirschner / FSFE e.V.
Schönhauser Allee 6/7
10119 Berlin
Germany

Contributions by: Gabriel Ku Wei Bin and Lucas Lasota.

Content of this report may be quoted or reproduced, provided that the source of information is acknowledged. All contents are, as long as not noted otherwise, licensed under CC BY-SA 4.0.

License information:
https://creativecommons.org/licenses/by-sa/4.0

Photo credit:
Adrien Olichon on Unsplash.

Acknowledgments: We would like to thank Dr. Till Jaeger and the Institut für Rechtsfragen der Freien und Open Source Software (ifrOSS: www.ifross.org) for their insightful contributions to the text.

Version: 1.2 – 05.2020

Table of Contents

CHOOSING A NAME FOR YOUR PROJECT

Select a good name for your project

When a user is searching for software to solve a problem they have, the first thing that they will encounter when they come across your project is its name. It is therefore important that you come up with a good name for your project.

An unfortunately named project can slow down its adoption if people do not take the name seriously, or if users have trouble remembering it. Some things to consider could be to give a name that would give users an idea of the project, or is related to what the project is used for, as well as one that would be easy to remember.

Make sure your project name is not already trademarked

Most importantly, it is absolutely essential that the name you choose is not the same as the name of some other pro<ject, and does not infringe upon any trademarks. The unauthorized use of a trademark (or a substantially similar mark) may result in a trademark infringement lawsuit being brought against you. For more information on how to check if your project name is already trademarked, see here.

Finally, you might wish to consider registering your project's name as a trademark with the European Union Intellectual Property Office (the "EUIPO"). While it can be costly to do so, it can ensure that your project name cannot be used by another person or entity for their project or business.

Summary

The goal here is to choose a project name that will not infringe on the existing rights of another project or company. In other words, a name that will not be confused with another entity's name for their software or related goods and services.

TRADEMARKS

What is a trademark?

A trademark is a sign (e.g., a name, a design, or even a colour) used by a person to distinguish their goods or services from those of their competitors. By recognizing the trademark, users or customers can identify which goods or services offered on the market originate from a particular trader.

Names and logos are the most common signs used as trademarks.

Why can't I use a name that is already trademarked?

Using a name or logo that is the same or substantially similar to an existing trademark is known as trademark infringement. Trademark infringement causes consumers to think a product or service is provided by a particular company, when it actually is not.

Laws against trademark infringement are put in place to protect both consumers and trademark owners against this type of confusing use of a trademark. They also serve to protect the trademark holder from potential damage to their reputation that can arise due to a difference in quality of the goods or services provided by the infringer.

In some cases, the same trademark may be registered and usable for different goods or services, if they are not well known and therefore would not cause confusion in the market. However, each case would be subject to a thorough investigation by the relevant authorities.

What are the consequences of trademark infringement?

Facing a lawsuit arising from trademark infringement would be costly, both in terms of payment of monetary damages to the original trademark holder, as well as the possible payment of court costs and attorney fees. Additionally, you would still need to come up with a new name at the end of the process.

It is therefore vital that you check if the name you intend to use for your project is not already taken, or substantially similar to, a registered trademark.

How do I check if my potential project name is already trademarked?

You can run a search for trademarks that could conflict with the name you have chosen with the EUIPO here and/or the WIPO Global Brand Database here. It does not cost anything to do a search on these websites.

Additionally, you can conduct online search engine searches, look in public source code repositories, and check domain names to see if there are any others who are using the same name as your project. One useful tool that you can also use is the "whohas" tool[1].

What do I get out of registering my project name as a trademark?

A European Union trademark will grant you the exclusive rights in all current and future Member States of the European Union. You can obtain these rights through a single registration. The trademark is valid for 10 years, and can be renewed indefinitely, for 10 years at a time.

Who will own the trademark if I register my project name?

Trademark rights can only be owned by a person, a group of persons or an entity. This is different from ownership of copyright in the code, which can be co-owned by community members who have contributed to the code.

Because only the owner of the project name may file for trademark registration, you must decide on who (whether an individual or an incorporated entity) will own the trademark rights. Nevertheless, it is possible for more than one person to apply for a trademark. In such a situation, the group of people would have to agree among themselves who would be permitted to enforce the trademark, as well as what would happen if someone wants to leave the project.

Here are some approaches you can take to decide who assumes ownership of your project's trademark rights:

a) The person who created the name remains the owner of the trademark, or transfers ownership to another individual;
b) A group of people, acting as partners or an unincorporated association owns the trademark; or
c) The person who created the name transfers ownership to an organization, society, or non-profit corporation.

It is best to outline an approach to the project name and any trademark rights, to avoid any potential conflict in the future. For example, the owner of a trademark can revoke the right of the project to use that trademark.

1 http://www.philippwesche.org/200811/whohas/intro.html

How do I register my project name as a trademark?

You can apply online for a European Union trademark online here. The EUIPO's online application form will guide you through the application process. Registering a trademark will cost between €850 and €1500. It is important to thouroughly outline the scope of the trademark protection in the list of goods and services for which the trademark shall apply. Please note that you have to actually use the trademark commercially; otherwise, any third party might apply for revocation of the trademark after 5 years of non-use.

Does a Free Software license cover the use of trademarks?

Differently from copyright and patents which protects your code, trademarks protects the users of your software. The idea of a trademark is to inform users/customers about the origin of your software. Therefore, trademarks denotes that the trademark owner is controlling the nature and quality of the goods or services sold under the mark.

With a Free Software license you inform the community how to use your software. However, most of Free Software licenses are focused only in copyright law and do not cover trademark rights. Some permissive liceses allow the usage of trademark, while some copyleft liceses don't. You can learn more about the trademark dispositions in Free Software licenses in our Free Software Licenses and Compatibility Matrix[2].

If you are incorporating code under trademark, contact the upstream to settle a permission for trademark use. Instead, if your project has trademarks, consider developing a trademark policy[3].

What is a trademark policy? How do I make one?

A trademark policy is a set of rules that inform others what is permitted and not to do with your trademark. The Free Software license of your project informs the community on how to use your software. However, most Free Software licenses focus only on copyright law, and do not cover trademark rights. A trademark policy can therefore enable you to better inform the community the guidelines of how you want your trademarks to be used, including explaining which icons or logo are your trademarks. If you choose to, you can state that they are not covered by the project license, and your trademarks can be licensed or used under the trademark policy.

You can learn more on how to implement a trademark policy with our Trademark Policy Guidelines[4].

---

2 The Free Software Licenses and Compatibility Matrix is available at https://download.fsfe.org/NGI0/.
3 For more information, please refer to our Trademark Policy Guidelines, available at https://download.fsfe.org/NGI0/.
4 The Trademark Policy Guidlines are available at https://download.fsfe.org/NGI0/.

Summary

Trademarks are important legal instruments for Free Software projects. A trademark represents the source of origin of a product, it also represents the quality of goods associated with the source. However, trademarks should not be mistaken for being *just* the project's name. Many Free Software projects set written guidelines for trademark use. For further reading: Guidelines for Trademark Policy.

FREE SOFTWARE AND LICENSES

The Basics

What is copyright?

Copyright is a legal construct that grants someone exclusive rights over a creative work. The most important exclusive right is in the name: the right to produce copies. Only the copyright holder is allowed to reproduce a work and to give new copies of their work to third parties. Additionally, the copyright holder has the exclusive right to modify the work and to make it publicly available (e.g. offering it up for download).

Usually, the author is the copyright owner, but copyright is often transferred to the author's employer, in many cases automatically. This is often stated in an employee's employment contract.

You do not need to do anything to gain copyright. As soon as you make a creative work, you (or your employer) inherently possess the copyright over it.

What is a license?

One problem with copyright (as it relates to software) is that it makes software unshareable by default. A license changes that. A license defines the terms under which the copyright holder allows the recipient of the license to use (i.e. to run, copy, distribute and/or modify) the software. If the license allows the recipient to use, study, share, and improve the software, then that software is Free Software.

How do I copy someone else's work?

If someone else has made their work available for you under a Free Software license, you can incorporate their work into your project.

When you put the work in one of the files in your project, you should add an *SPDX-FileCopyrightText* tag for the copyright holder(s) and an *SPDX-License-Identifier* tag for the license(s) under which the work was made available. See the REUSE section for more information.

If the work was licensed differently from your project, you should verify whether the licenses are compatible, and add the new licenses to your project.

Can I copy a work that has no copyright notice or license?

Before you proceed, always first make sure that you can find the copyright and licensing information elsewhere. Some projects only include this information in the root directory or in their README file.

If the work has no license, that means that you do not have the right to copy it, or it is in public domain. If you believe that this is a mistake and the author clearly meant for you to be able to copy this work, you should contact the author and ask them to license their work. Feel free to refer them to https://reuse.software.

Which files are copyrightable?

All files that are original works of authorship are copyrightable. In essence. If someone sat down typing their own original thoughts on a keyboard, then that person holds copyright over the output. Common examples are source code, documentation, audio, and video.

However, there are some edge cases. For example, the program

```
print("Hello, NGI!")
```

probably does not meet the threshold of originality because it is too trivial. Similarly, data files and configuration files may not meet that threshold either.

What is a copyright holder, and what is an author?

In these resources, we maintain a distinction between the copyright holder and the author. The author (also known as creator) is the person who sat down and created a work. Think of the author as a programmer, writer, or artist.

The copyright holder is the person who has the exclusive rights over that work. Often the author and the copyright holder are the same. However, if the author is being paid by their employer to create a work, the employer is the copyright holder.

Keep in mind that in some jurisdictions, the word "author" is often used as a synonym for "copyright holder". In other jurisdictions, authors maintain some rights over their work even if they are not the copyright holder.

What is public domain?

The public domain consists of all the creative work (including software) to which no copyright applies. The rights to these works may have expired, been expressly waived, or may be inapplicable. As rights vary by country and jurisdiction, the Creative Commons CC0 1.0 Universal (CC0) License can be used to waive copyright to the extent you may have these rights in your work.

Please note that the absence of a license does not make a software free to use. Software is copyrighted by default, so unless it has been explicitly and validly placed in the public domain, using code without a license may be considered copyright infringement.

What is Free Software?

The term “Free Software” (also called Open Source Software) is a regulated term created by the Free Software Foundation. It refers to software that enables its users to maintain their control and freedom over how to use such software. More specifically, the following four essential freedoms define Free Software:

- **Freedom to Use**

  Installing and running the program is not restricted. Placing restrictions on the use of Free Software, such as time ("30 days trial period", "license expires January 1st, 2004"), purpose ("permission granted for research and non-commercial use", "may not be used for benchmarking"), or geographic area ("must not be used in country X") makes a program non-free.

- **Freedom to Study**

  Placing legal or practical restrictions on the comprehension or modification of a program, such as mandatory purchase of special licenses, signing of a Non-Disclosure-Agreement (NDA) or - for programming languages that have multiple forms or representation - making the preferred human way of comprehending and editing a program ("source code") inaccessible also makes it proprietary (non-free). Without the freedom to modify a program, people will remain at the mercy of a single vendor.

- **Freedom to Share**

  Software can be copied/distributed at virtually no cost (i.e. license fees and royalties are not permitted). If you are not allowed to give a program to a person in need, that makes a program non-free. This can be done for a charge, if you so choose.

- **Freedom to Improve**

  Not everyone is an equally good programmer in all fields. Some people don't know how to program at all. This freedom allows those who do not have the time or skills to solve a problem to indirectly access the freedom to modify. This can be done for a charge.

These freedoms are rights, not obligations, although respecting those freedoms for society may at times oblige the individual. Any person can choose to not make use of them but may also choose to make use of all of them.

Free Software does not exclude commercial use. If a program fails to allow commercial use and commercial distribution, it is not Free Software. A growing number of companies base their

business model completely, or at least partially on Free Software, including some of the largest proprietary software vendors. Free Software makes it legal to provide help and assistance; it does not make it mandatory to do so.

Free Software is therefore a matter of liberty, not of price; the word “free” in Free Software means “free” as in “free speech”, not as in “free beer”. You can learn more about the concept of Free Software here. You can also refer to a list of approved Free Software licenses here.

What is Open Source software?

You might have also come across the term “Open Source” software. Originally intended as a synonym for Free Software, the term “Open Source” is the other major term connected to software that also describes the complete set and range of software licenses that give users the right to use, study, share, and improve the software. The term is regulated by the Open Source Initiative (the OSI), which states that Open Source doesn't just mean access to the source code. The distribution terms of open-source software must comply with the following criteria:

1. Free Redistribution: The license shall not require a royalty or other fee for the distribution by the licensee.
2. Source Code must be included
3. Derived Works must be permitted
4. Integrity of The Author's Source Code: The license may require derived works to carry a different name or version number from the original software.
5. No Discrimination Against Persons or Groups
6. No Discrimination Against Fields of Endeavor
7. Distribution of License: The rights attached to the program must apply to all to whom the program is redistributed without the need for execution of an additional license by those parties.
8. License Must Not Be Specific to a Product
9. License Must Not Restrict Other Software
10. License Must Be Technology-Neutral

You can refer to a list of approved Open Source licenses here.

What is proprietary software?

Proprietary software is the opposite of Free Software. This term is used to refer to software that is distributed under restrictions that prevent users from enjoying the four essential freedoms. Even software that is distributed gratis can be considered proprietary, if its restricts users from any of the four essential freedoms.

The term “commercial software” is sometimes used carelessly to refer to proprietary software. This would be incorrect: as explained below, Free Software does not exclude commercial use.

How can Free Software be used commercially?

Free Software can be sold, as long as the buyers are not restricted from sharing copies of the software after their purchase. Hence, requiring a fee for any copy distributed by the user is not permitted. Free Software can also be commercialized in other ways, such as by selling support, services, and certification. Free Software remains free as long as the four essential freedoms outlined above are afforded to users.

Why is it important that your project uses/produces Free Software?

Software is the most important cultural technology of the 21st Century; it is almost impossible to imagine daily life today without it. When others control a tool as important to society as software, they can exert great influence over our lives and actions.

For example, whoever controls the search engines that we use has the power to determine what we find online; whoever controls the software on which our online transactions are run can have access to our personal data. It would be dangerous to democracy if the critical social instrument that software constitutes were controlled by only a small group.

The freedoms in Free Software to use, study, share, and improve can therefore be applied to hand control of the future of software back to the people. In line with the Next Generation Internet's mission of creating an internet that respects human and societal values, having your project use and be Free Software would help to foster an open and transparent internet; one where users are able to share and improve the tools that it provides.

Software Licensing

What is a software license?

Generally, a license is an authorization to use, release, or distribute someone else's property. This includes intellectual property such as a piece of text, a song, or software.

Software licenses tell people how the rights holder (usually the author of the software) wants the software to be used and what freedoms, or restrictions, the software has. Without a license, any use of the software is prohibited by default except in the case of a statutory exception to copyright[5].

Please note that the absence of a license does not make a software free to use. Software is copyrighted by default, so unless it has been explicitly and validly placed in the public domain, using code without a license may be considered copyright infringement.

5 Articles 5 and 6 of Directive 91/250/EEC 2009/24/EC on the legal protection of computer programs (the Computer Programs Directive) provide such statutory exceptions. For more information, see the full text of the Directive here.

Are there different types of software licenses?

Yes. Software licenses can be classified according to the rights granted or retained by the copyright owner. When the owner of a software waives all of these rights, the software is in the Public Domain. Please note that a complete waiver of copyright may not work in all jurisdictions. For such cases, use of the CC0 License achieves this goal practically.

If a software is under a proprietary license, only a few rights are granted; for example, the right to use program but not to distribute copies of it. If the license grants the four essential freedoms to use, study, share and improve the software, it is a Free Software license. You can find here lists of approved Free Software licenses from the Free Software Foundation and Open Source Initiative. For other license categories, you can refer to this list prepared by IfrOSS here.

The following diagram illustrates the type of licenses and how many rights they would grant to the user or retain for the copyright owner.

Rights granted or retained in software licensing

Public Domain	Permissive Licenses	Copyleft Licenses	Proprietary Licenses	Trade Secret
All rights waived	<<< More rights granted to user		>>> More rights retained for owner	All rights retained

Can I use a software license for other types of works?

Although Free Software licenses can be used for non-software works, there are licenses that suit better other purposes, like documentation, database or hardware.

Licenses like the CC BY 3.0 (Attribution), CC-BY-4.0 (Share Alike) or Gnu Free Documentation License can be adequate for documentation purposes. You can find a list of licenses for non-software works here. Please note that if you use different licenses for your software and its documentation, source code examples in the documentation are also licensed under the software license.

If you want to waive your copyright rights over your software and put it under public domain, consider the CC0 license.

You can find examples of Open Data and Open Hardware licenses here.

What is a Free Software license?

Please refer to the section What is Free Software?

What is an Open Source license?

Please refer to the section What is Open Source Software?.

What is copyleft?

Copyleft is a form of Free Software licensing that gives permission for a user to enjoy the four freedoms of Free Software, under the condition that those freedoms remain intact in further distribution of the software or derivative works. That's why copyleft licenses are also called "reciprocal licenses"[6]. This is in contrast to "permissive or non-reciprocal" licenses, which, unlike copyleft licenses, allow the creation of derivative versions of the software with different license terms from the original, which can in some cases result in a derivative work being placed under a proprietary license.

In practical terms, having a copyleft license would generally mean that the derivative and/or combined work has to be licensed under the same license as the original work. As a result, this ensures that a copylefted piece of Free Software can remain free, and cannot be transformed into proprietary software.

An example of a copyleft license is the GNU General Public License. Under this license, derivative works can only be distributed under the same license terms. This is distinct from permissive (non-reciprocal) licenses, which include the commonly used MIT License and Apache License 2.0.

For more information, see Basic Conditions in Software Licensing in the summary at the end of this section.

What is weak and strong copyleft?

Free Software licenses embody the norms of the communities that use them, in such a way that the reciprocity of these norms is to be expected. Reciprocity in this case means that consistent licensing must be maintained. Nevertheless, different copyleft licenses can impose different sets of obligations, and this can result in varying degrees of protections over the original work, from strong protections, to weaker ones.

In its strongest form, copyleft can place conditions on the licensing of all the other code compiled together to make the eventual binary executable program, and require that all modifications of a program (i.e. any derivative work) have to be licensed under the original license. Licenses like the GPL set the scope of the expected reciprocity to submit under copyleft any code needed to create the resulting project binary. In this case the license is "project-scoped reciprocal".

6 Read more: *Permissive and Copyleft Are Not Antonyms, 2017.* https://opensource.org/node/875

Weaker copyleft licenses have a more defined scope of conditions, and set the scope of the expected reciprocity to the individual source files within the project, rather than the whole project collectively. If you change a file that is part of the project, or reuse code from a file in the project in a new codebase, the resulting file must be licensed the same way as the original file, but there are no requirements placed on other files combined together to create the program. Therefore, these licenses can be called “file-scoped reciprocal licenses.”

An example of what constitutes weak copyleft would be the Mozilla Public License, version 2.0 (the MPL 2.0), which has a file based copyleft. This means that the copyleft obligations is limited to only those source code files that contain code licensed under the MPL 2.0. Similarly, the GNU Lesser General Public License version 3.0 (LGPL 3.0) limits copyleft obligations to a library, but allows linking with components under different license terms.

For more information, see Basic Conditions in Software Licensing in the summary at the end of this section.

What is network copyleft?

Copyleft licenses allow the essential freedoms of using, reproducing, adapting, or distributing under the condition that those freedoms remain intact in further distribution of the software or its derivative works.

In general terms, the trigger of the copyleft conditions is the distributing of the software. However, for programs offered as services over a network, the traditional distribution concept becomes blurry.

Therefore, network copyleft licenses (AGPL for example) ensure that the source code is available to users of network, triggering copyleft not only by distributing but also by using it.

For more information, see Basic Conditions in Software Licensing in the summary at the end of this section.

What is a permissive license?

A “permissive license” refers to a Free Software license with minimal requirements about how the software and its derivative works can be redistributed. Licenses like the MIT License and Apache License 2.0 do not impose a condition that any derivative works must be distributed under the same copyleft license terms.

Because permissive licenses do not impose conditions on derivative works, in contrast with copyleft licenses, permissive licenses therefore permit the licensed code to be used in proprietary derivative works. They are also known as non-reciprocal licenses[7].

For more information, see Basic Conditions in Software Licensing in the summary at the end of this section.

7 Read more: *Permissive and Copyleft Are Not Antonyms,* 2017. https://opensource.org/node/875

Summary

A Free Software license is an important instrument used to hand control of your project's software back to the people, allowing them to use, study, share, and improve the code. The table below illustrates some basic information about the conditions of Free Software licenses.

Basic conditions in software licensing[8]		
Permissive licenses	**Weak copyleft licenses**	**Strong copyleft licenses**
Examples: MIT, Apache 2.0. etc	Examples: LGPL, MPL, etc	Examples: GPL, AGPL, etc
. No restriction to code's use . (Sometimes) copyright notice requirement	. No restriction to code's use . Copyright notice requirement . Along the binaries, source code must be avaiable . Source code must be avaiable under the same copyleft terms . No restrictions on the exercise of the license . Combined code may be not submitted to copyleft	. No restriction to code's use . Copyright notice requirement . Along the binaries, source code must be avaiable . Source code must be avaiable under the same copyleft terms . No restrictions on the exercise of the license . Combined code may be submitted to copyleft

8 Summary based on Meeker, Heather. *Open Source for Business.* 2.ed. Fleming, 2017. p. 33, 34.

CHOOSING A FREE SOFTWARE LICENSE

There are many Free Software licenses out there for you to choose from for your project. A large number have been written to satisfy the particular legal needs of a corporation or person, and in all likelihood would not be appropriate for your project. In most cases, it would be best for you to take a look at the most commonly used Free Software licenses, and to choose one from there that most suits your needs. This reduces the risk of license incompatibilities.

In this section, we will help you understand the importance of ensuring that the licenses in your project are compatible, and provide guidance on how to choose a license for your project's needs.

Once you have decided on which Free Software license to use for your project, you will then need to apply it to the software. The final part of this section will provide you guidance on how to do so.

How to choose a Free Software license

Can I write my own Free Software license for my work?

Anyone can write their own Free Software license to dictate the terms and conditions of use of their work. However, we strongly recommend you to use an existing well-known license, instead of making up your own.

Licenses should be written to apply in complex legal issues, and people are more likely to use works under established and existing licenses whose legal effects are well known and have been clearly documented. The legal effects of a new license would be ambiguous, and this would have the effect of discouraging users from adopting your software.

Another reason to not write your own license is to prevent license proliferation. There are already numerous licenses available with very similar terms, which are often incompatible with each other. The standardization of license terms can make things easier for everyone. Conversely, inventing a new license would make it harder for the community to keep track of all available licenses and would be a hindrance to developers who simply want to combine code under different licenses.

You can refer to our Free Software Licenses and Compatibility Matrix[9] for an overview of the most common Free Software licenses.

9 The Free Software Licenses and Compatibility Matrix is available at https://download.fsfe.org/NGI0/.

What should I consider when choosing a license?

Having a Free Software license is the best way to protect your project's contributors and users. A license can determine the way people contribute to the project and how the software will be used and shared.

If you are taking part in a project that has already determined which licenses to use, the best way to contribute is to continue using that project's license. Some communities have strong preferences for particular licenses. If you want to participate in one of these communities, it will be easier to use the preferred license even if you're starting a brand new project with no existing dependencies. However, if you don't see your project as part of a community, you have more flexibility with which Free Software license to choose.

There are two objectives to pursue with a Free Software license:

a) Make the code as simple and permissive as possible. If you plan to let people and organisations to use your code in the most simple way, permissive licenses let people do almost anything they want with your project, including to make and distribute closed source versions.
b) Protecting the code's freedom. Copyleft protects software freedom, permitting people to do almost anything they want with your project, *except* to distribute proprietary versions.

For more information the terms and conditions of permissive and copyleft licenses, you can refer to our Free Software Licenses and Compatibility Matrix.

What are some of the most common licenses that I can use for my project?

The following table[10] lists some of the most common Free Software licenses, in order of the most permissive to the least.

10 Table based on the following works: https://choosealicense.com/licenses/ and Wheeler, David. *The Free-Libre / Open Source Software (FLOSS) License Slide*, 2017. Avaiable at: https://dwheeler.com/essays/floss-license-slide.html

Permissive	Weak copyleft	Strong copyleft licenses	Network Copyleft
Allow any kind of usage of code. Permissive licenses permit software to become proprietary.	***Prevent the software component (often a software library) from becoming proprietary, yet permit it to be part of a larger proprietary program.***	***Prevent the software from becoming proprietary.***	***Prevent the software from becoming proprietary on network enviroment, triggerring copyleft not only by distributing but also using it.***
Most popular liceses (SPDX code)			
MIT (MIT) A short and simple permissive license with conditions only requiring preservation of copyright and license notices. Licensed works, modifications, and larger works may be distributed under different terms and without source code.	**LGPL (LGPL-3.0)** The license allows developers and companies to use and integrate a software component released under the LGPL into their own (even proprietary) software without being required by the terms of a strong copyleft license to release the source code of their own components. The use of this license is discouraged by the FSF.	**GPL v3 (GPL-3.0)** The most popular Free Software license. Permissions of this strong copyleft license are conditioned on making available complete source code of licensed works and modifications, which include larger works using a licensed work, under the same license. Copyright and license notices must be preserved. Contributors provide an express grant of patent rights.	**Affero GPL v3 (AGPL-3.0)** Based on GPL v3, is dedicated to software used over a network (SaaS for instance). This provision requires that the full source code be made available to any network user.
Apache 2.0 (Apache-2.0) A permissive license whose main conditions require preservation of copyright and license notices. Contributors provide an express grant of patent rights. Licensed works, modifications, and larger works may be distributed under different terms and without source code.	**Mozilla Public License 2.0 (MPL 2.0)** A weak copyleft license, characterized as a middle ground between permissive free software licenses and the GNU General Public License (GPL), that seeks to balance the concerns of proprietary and open source developers. a larger work using the licensed work may be distributed under different terms and without source code for files added in the larger work.		

For more information the terms and conditions of permissive and copyleft licenses, you can refer to our Free Software Licenses and Compatibility Matrix.

What license should I use for my software project?

For the purposes of the Next Generation Initiative, we recommend that you use the GNU General Public License v3 (**GPL-3.0)** for your software project.

However, this is not a one-size-fits-all recommendation. You should consider the nature of your project, and decide which of the licenses above would fit your needs.

For more information the terms and conditions of permissive and copyleft licenses, you can refer to our Free Software Licenses and Compatibility Matrix.

What is an MIT license? When should I use it?

The MIT License is a permissive license, that only requires the preservation of copyright and license notices. It permits users to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the licensed software without any restrictions. This means that any derivative works can be distributed under different terms and without access to the source code, and can be made into proprietary software.

If you are comfortable with your project's code potentially being used in proprietary programs, you can choose to use the MIT license.

What is the GPL? When should I use it?

The GNU General Public License v3 (GPL-3.0) is a copyleft license that is probably the most widely recognized Free Software license available. This is in itself a big advantage, since many potential users and contributors will be familiar with it, and therefore won't have to spend extra time to read and understand your license.

The GPL is the also the most popular copyleft license. If you would like your code to be able to be merged freely with code that is covered by the GPL, then you should choose the GPL, or a GPL compatible license.

The Free Software Foundation in the United States of America maintains a list showing which licenses are compatible with the GPL (as well as those that are not) here.

What is the LGPL? When should I use it?

The GNU Lesser General Public License version 3 (LGPL-3.0) license is known as a weak copyleft license because it permits linking with non-free modules, allowing the use and integration of a software component into a derivative or combined work (that can even be proprietary) without the requirement to release the source code of their own components.

Nevertheless, any derivative work with an LGPL-covered component is required to make their modified version available under the same LGPL license.

The LGPL is named as a “Lesser” GPL license, because it does not guarantee the four freedoms of Free Software for the use of the software as a whole. Rather, it only guarantees the freedom of modification for components licensed under the LGPL. Because of this, we do not generally recommend the use of the LGPL.

What is the Apache License? When should I use it?

The Apache License version 2.0 (Apache 2.0) is a permissive free software license written by the Apache Software Foundation. This license allows users to use, distribute, modify, and distribute modified versions of a licensed software for any purpose.

As a permissive license and not a copyleft license, the Apache license does not require a derivative work of the software, or modifications to the original, to be distributed using the same license terms. Because of this, derivative or modified works of Apache licensed software can be be distributed under different terms and without source code. This also means that derivative works may be proprietary software.

Summary

Choosing a Free Software license for your project is one of the first steps to be made. We listed several licenses that fit a broad spectrum of purposes for you to choose for your project. It is fundamental to understand how pieces of software under different licenses interact in the final product.

Applying a Free Software license

Once you have decided on which Free Software license to use for your project, you will then need to apply it to your software. This section will provide you guidance on how to do so, as well as best practices that you are encouraged to adopt.

Applying a Free Software license to your software

What steps should I take to ensure the Free Software license is applied to my software?

To apply the license to your software, you will need to do 3 things: first to identify all copyright holders and reach an agreement for a license, second to effectively inform the public which license you intend the software to be released under, and third to ensure that the software itself includes the license.

Step 1: Identify all copyright holders and seek consent for the intended license

The licensing over your software project is invalid, if not all copyright holders agree upon such licensing. Hence, the first step is to identify all copyright holders in your software project, and to ensure that all of them agree upon a license.

Step 2: Informing the public of your intended license

You should first state which license you are using clearly on the project's front page. It is also recommended that you put a copy of the license text somewhere on the project's website, or GitHub page.

Step 3: Including the license in your software

You can include the license in your software by putting the full license text in a file called COPYING (or LICENSE) included with the source code. You should also place a short notice in a comment at the top of each source file, naming the copyright date, holder, and license. This notice should also include a note on where to find the full text of the license.

What would a notice in my source files look like?

In general, the notices that are placed in source files do not have to look exactly like each other, and can take many different forms, as long as the standard requirements are met. However, we recommend adopting the REUSE specifications, which can simplify your work and foster the integration and reuse of your software through standardizing how notices are displayed.

‘Standardizing notices also helps automated analysis of all applicable licenses. See the REUSE section for more information.

Software Package Data Exchange (SPDX)

What is SPDX?

SPDX is a project by the Linux Foundation and stands for Software Package Data Exchange. It is an open standard for communicating components, licenses, and copyrights associated with a particular version of software packages. The need to identify the license for Free Software is critical for both reporting purposes and license compliance. You can learn more here.

SPDX reduces redundant work by providing a common format for companies and communities to share important data about software licenses, copyrights, and security references. SPDX provides a common language and vocabulary to form a precise and unambiguous software bill of material (SBOM). It also creates a simple format for licenses that is machine-readable to help automation and is short enough to help humans quickly handle common cases. In order to use SPDX it is necessary to understand three basic concepts[11]:

- SPDX license identifiers;
- SPDX license expressions; and
- SPDX files.

What is an SPDX license identifier?

An SPDX License Identifier is a human readable short text string that uniquely identifies a license, or in other words, a standardized shortened version of the license name.

The SPDX License List can give you the commonly found licenses and exceptions used in Free Software, and other collaborative software or documentation. You can also refer to the REUSE section of these documents to learn how to apply the SPDX License Identifiers into projects.

What are SPDX License Expressions?

In some situations where there isn’t just one license, for example where software might be offered with a choice between two or more licenses, SPDX allows you to clearly express choices when you need to. You can use the following different operators to combine license expressions to create other license expressions:

- "OR": Recipients can choose between two or more licenses.
- "+": This means "this license or any later version".
- "AND": Recipients are required to simultaneously comply with two or more licenses.

11 This material is based on David Wheeler‘s tutorial on SPDX: https://github.com/david-a-wheeler/spdx-tutorial.

- "WITH": Add the following named exception. For example, "(GPL-3.0-or-later WITH Classpath-exception-2.0)" means recipients must comply with the GPL version 3.0 or later with the Classpath 2.0 license exception.

Most programming languages have a language library package manager, and most systems have a system-level package manager. Each package manager has a package format. In almost all of them there is a “license” field, and in most cases you can simply use a SPDX license expression in that field.

What are SPDX files?

SPDX files are a way to capture and exchange license information in a way that is independent of the programming languages and package managers used.

The SPDX specification actually supports the *tag:value* format. A *tag:value* file is normally a sequence of lines, where each line is a tag name, a colon, a space, and its value. There are many tags defined in the SPDX specification. You can learn more about this on the SPDX website.

What are some examples of SPDX tags?

Here are some especially important or useful tags:

- SPDXVersion: The version of the spec used, normally "SPDX-2.1".
- DataLicense: The license for the license data itself.
- Creator: Who or what created this SPDX file (not the package creator). This is in one of 3 formats:
  - For a person: person name, optionally followed by email in parentheses.
  - For an organization: organization name, optionally followed by email in parentheses.
  - For a tool: toolidentifier-version
- PackageName: The full name of the package as given by Package Originator.
- PackageOriginator: The person or organization from whom the package originally came.
- PackageVersion: The version number of this particular version of the package (optional).
- PackageHomePage: The package's home page URL.
- PackageLicenseDeclared: The license identified in text in one or more files (for example a COPYING or LICENSE file) in the source code package. This field is not intended to capture license information obtained from an external source, such as the package website.

An example of a package expressed as a *tag:value* is:

```
SPDXVersion: SPDX-2.1
DataLicense: CC0-1.0
```

PackageName: Foo
PackageOriginator: David A. Wheeler
PackageHomePage: https://github.com/david-a-wheeler/spdx-tutorial/
PackageLicenseDeclared: MIT

We recommend that you use ".spdx" for an SPDX file in the *tag:value* format (i.e. "LICENSE.spdx").

If you are developing software or other copyrightable content, you will still need to select a license and express it in a manner that humans can understand. For software, create a file named LICENSE or COPYING (possibly with a .md or .txt extension) to provide human readable text of the license.

Also create an SPDX file, as described above, so programs can automatically process exactly what the license is, and humans can have a short and precise way of representing the license of the software.

The REUSE Initiative

What is the REUSE Initiative?

The REUSE Initiative (or simply, "REUSE"), is an project of the Free Software Foundation Europe to develop awareness of the best practices for expressing license and copyright information in Free Software projects. It intends to facilitate management of source code by making licensing and copyright information more consistent in how it is added to source code in ways which allow for automating many of the processes involved.

How do I make my project REUSE compliant?

Making your project REUSE compliant can be done in three steps:

- Choose and provide licenses
- Add copyright and license information to each file
- Confirm REUSE compliance

You can find more information in a later section in this FAQ on REUSE.

Summary

SPDX and REUSE represent the best practices that you can follow in order to apply a Free Software license to your project.

REUSE FAQ

The REUSE Initiative (or simply, “REUSE”), is a project of the Free Software Foundation Europe to develop awareness of the best practices for expressing license and copyright information in Free Software projects. It intends to facilitate management of source code by making licensing and copyright information more consistent in how it is added to source code in ways which allow for automating many of the processes involved.

This section will show you the basic methods to make a software project REUSE compliance. By the end of this document, all of your files should clearly have their copyright and licensing marked, and you will be able to verify this using a linter tool.

Making your project REUSE compliant can be done in three simple steps:

- Choose and provide licenses
- Add copyright and license information to each file
- Confirm REUSE compliance

You can use the following tutorial to do so. For the purposes of this tutorial, we will assume that you have chosen to use the GNU General Public License v3.0 or any later version (the “GPL”), and that the directory of your project looks like this:

```
project/
├── img/
│   ├── cat.jpg
│   └── dog.jpg
├── src/
│   └── main.c
├── .gitignore
├── Makefile
└── README.md
```

Choosing and Providing Licenses

How do I provide licenses?

After choosing a license, you will need to put the license in your project directory. You can find your license in the SPDX License List. SPDX is an open standard for communicating license and copyright information. Each license is uniquely identified by a shortform SPDX License Identifier. The SPDX License Identifier for your chosen license is GPL-3.0-or-later.

You create a LICENSES directory in your project root which will contain all the licenses that you use in your project. You then download your license from the *license-list-data* repository and put it in the *LICENSES* directory.

Adding Copyright and Licensing Information

How do I add copyright and license information to each file?

Now that you have a license, you need to indicate in the relevant files that these files fall under that license. You edit the comment header of *src/main.c* as such:

```
/*
 * SPDX-FileCopyrightText: 2019 Jane Doe <jane@example.com>
 *
 * SPDX-License-Identifier: GPL-3.0-or-later
 */
```

The *SPDX-FileCopyrightText* tag records the publication years and the copyright holder of the contents of the file.

The *SPDX-License-Identifier* tag is followed by a valid SPDX License Expression, which is typically just the SPDX License Identifier of the license.

You can be flexible with the format, just make sure that the line starts with *SPDX-FileCopyrightText:*.

Each file must always contain these two tags in the header. You are allowed to use the tags multiple times if you have multiple copyright holders or licenses.

In the example project, you can also edit *Makefile* and *README.md* using this header information, but of course with corresponding comment syntax.

How do I deal with a file that has been edited by many people?

Some files are edited by many people and would have an extremely long list of copyright holders in the header. This may be aesthetically unpleasing or even impractical, but is not incorrect.

If you would rather not deal with having so many copyright notices, some projects such as Chromium circumvent this problem by using the pseudonym "Copyright (c) 2013 The Chromium Authors" as their copyright tag. You may consider doing this if all copyright holders and authors agree to do so, but then you should keep a list of copyright holders and authors in a separate file in your project.

Why can’t I use version control to record copyright?

As previously explained, there are differences between the copyright holder and the author, which do not always refer to the same person. Version control typically only records authorship, which makes it unsuitable for the task of recording copyright.

Another obstacle is that version control history may contain errors, and fixing such an error would require rewriting the history, causing all contributors to have to re-download the new trunk.

A further issue with version control is that the *blame* command that is typically used to find authorship line-by-line shows only the author of the last commit in that line, even if it was a trivial commit such as fixing a typo.

Is there a standard format for copyright notices?

Generally, we recommend that you use the following format:

> *SPDX-FileCopyrightText: [year] [copyright holder] <[contact address]>*

You may choose to drop items except the copyright holder, which must always be included. We recommend that you include all items, however. For more information, check our SPDX section.

Do I use SPDX-FileCopyrightText, Copyright, or ©?

The following copyright notices are valid examples that you can use for your copyright notices:

> *SPDX-FileCopyrightText: 2019 Jane Doe <jane@example.com>*
> *SPDX-FileCopyrightText: © 2019 John Doe <joe@example.com>*
> *© Example Corporation <https://corp.example.com>*
> *Copyright 2016, 2018-2019 Joe Anybody*
> *Copyright (c) Alice*

Out of the above, we highly recommend using the first two. The others exist primarily to be compatible with existing conventions.

What years do I include in the copyright statement?

Generally, there are four options for you to choose:

- The year of initial publication
- The year of the latest publication
- All years of publications, either as range (e.g. 2017 – 2019) or as separate entries (e.g. 2017, 2018, 2019).
- Do not include any year

Which option you choose is ultimately up to you.

Where else do I put my license information?

Marking all individual files with *SPDX-License-Identifier* tags goes a long way towards unambiguously communicating the license information of your project, but it helps to communicate the license information in natural language as well.

In the README of your project, feel free to provide a summary of the licensing information, or simply redirect the reader to your *LICENSES/* directory.

Additionally, many package hosting sites expect that you declare the licensing information of your package.

What are license exceptions and what do I do with them?

License exceptions are additions or alterations to a license that often work to permit a certain use of the code that wouldn't be allowed under the original license. It is often used by compilers, where a portion of compiler code may end up in the resulting binary. The exception may waive rights over portions of code that end up in binaries.

Exceptions are treated almost identically to licenses. You can combine a license with an exception by marking a file with the following tag:

> *SPDX-License-Identifier: GPL-3.0-or-later WITH GCC-exception-3.1*

What do I do with binary and uncommentable files?

You will also want to license your image files under GPL-3.0-or-later. Unfortunately, images and other binary files do not have comment headers that one can easily edit. Other examples include automatically generated files and certain data and configuration files for which comments are non-trivial.

There is a simple trick to circumvent this. Create the files *cat.jpg.license* and *dog.jpg.license*, each containing the same information about license and copyright holder as above.

What do I do if there are files that have a different license?

It may arise that the photos of the cat and the dog were not licensed under the GPL at all, but under the Creative Commons Attribution 4.0 International, and owned by John Doe. If this happens, then you should check the SPDX License Identifier of the license. In this case, it is *CC-BY-4.0.*

You should then create the file *LICENSES/CC-BY-4.0.txt*, following the same steps you used for GPL-3.0-or-later.

You can then edit *cat.jpg.license* and *dog.jpg.license* to say:

> *SPDX-FileCopyrightText: 2019 John Doe <john@doe.com>*
>
> *SPDX-License-Identifier: CC-BY-4.0*

Nevertheless, icons, images, and other non-code files are considered separate works from the code itself. While this section explains what to do with these non-code files in order to make your repository REUSE compliant, you should nevertheless not be concerned if these files are licensed under a different license from your code.

Do I need to provide licensing information for build artifacts?

When you compile your program, you will generate some build artifacts, such as *src/main.o*. You do not need to provide any licensing information for those files. Just use your *.gitignore* file to ignore these build artifacts. The REUSE tool will respect the contents of *.gitignore*.

What do I do with insignificant files?

You probably will have files in your project that you do not find particularly copyrightable, for example configuration files such as *.gitignore*. Intuitively you may not want to license these files, but the fundamental idea of REUSE is that all your files will clearly have their copyright and licensing marked.

One way to indicate that you do not exercise any copyright over this file is by marking it as being in the public domain. Independently from the question of whether or not the file contains copyrightable material, this avoids unclear licensing and interpretation issues. The CC0 license is a good way to do this. Edit the file to contain:

```
# SPDX-FileCopyrightText: 2019 Jane Doe <jane@example.com>
#
# SPDX-License-Identifier: CC0-1.0
```

Resulting Project Tree.

Your project tree will now look like this:

```
project/
├── img/
│   ├── cat.jpg
│   ├── cat.jpg.license
│   ├── dog.jpg
│   └── dog.jpg.license
├── LICENSES/
│   ├── CC0-1.0.txt
│   ├── CC-BY-4.0.txt
│   └── GPL-3.0-or-later.txt
├── src/
│   └── main.c
├── .gitignore
├── Makefile
└── README.md
```

Confirming REUSE Compliance

How can I confirm if my project is in compliance with REUSE guidelines?

Now that you have marked all files with their copyright and licensing, it is time to check whether you did not miss anything. To do this, we provide a linter tool for you to use. You can read the full documentation here, or read the quick steps below.

```
$ # Install the dependencies for the tool.
$ sudo apt install python3 python3-pip
$
$ # Install the tool
$ pip3 install --user fsfe-reuse
```

The executable is now in *$HOME/.local/bin/*. Make sure that this is in your *$PATH*. Now go to the project directory and run the linter.

```
$ cd path/to/project/
$ reuse lint
SUMMARY

Bad licenses: 0
Missing licenses: 0
Unused licenses: 0
```

Used licenses: CC-BY-4.0, CC0-1.0, GPL-3.0-or-later
Read errors: 0
Files with copyright information: 6 / 6
Files with license information: 6 / 6

Congratulations! Your project is REUSE compliant :-)

As you can see in the last line, the tool confirms that your project is compliant with REUSE now! To learn what the different numbers mean, please have a look at the full documentation of the reuse tool.

Summary

REUSE is part of the best practices to comply with Free Software licenses copyright and license notices requirements. This section will show you the basic methods to make a software project REUSE compliant. You will learn how to have your code properly display your copyright and licensing, and verify this using a linter tool.

LICENSE COMPATIBILITY AND THIRD PARTY CODE

If you intend to incorporate or merge existing code into your software project, it is important that you ensure that the licenses that cover those existing code are compatible with each other, and with the license that you intend to issue your software project under.

Compatibility of Free Software Licenses

What does it mean to say that licenses are compatible?

When your project combines two pieces of Free Software into one, or merges code from one into another, it is important to pay attention to whether the licenses of each software or code allow this combination, or prohibit it. If the licenses allow it, the licenses can be said to be “compatible” with one another. If they prohibit it, the licenses can be said to be “incompatible” with one another.

In other words, we say that several licenses are compatible with each other if if it possible to combine code under these different licenses, while still complying with the terms and conditions of all of these licenses. It is therefore important that, when you incorporate existing code from a third party, you ensure that the license of that third party code is compatible with the license that you intend to issue your software project under.

Being compatible also means that code under one Free Software license can be combined with code under another, and the resulting software can be distributed under either Free Software license without violating the terms of the other.

Using incompatible licenses may result in copyright infringement.

Can code covered by Free Software licenses be used in proprietary software?

Generally, Free Software licenses allow anyone to modify the code covered, and to redistribute it in both its original and modified form.

Permissive Free Software licenses (i.e. non-copyleft licenses) allow their code to be used in proprietary derivative works. In such a case, the derivative work will still be proprietary, not withstanding that it contains code from a source that is covered by a Free Software license. Examples of permissive licenses that allow covered code to be used in proprietary derivative works include the Apache license, the MIT license, and BSD licenses. Conversely, copyleft

licenses like the General Public License (GPL) requires that any derivative work must be distributed under the same terms, and be Free Software as well.

Are all Free Software licenses compatible with each other?

No. While all of the commonly-used permissive Free Software licenses are compatible with each other, copyleft licenses (that do not allow their covered code to be used in proprietary software) can and are prone to creating license incompatibilities, depending on each unique situation and the licenses involved. For more information the terms and conditions of permissive and copyleft licenses, you can refer to our Free Software Licenses and Compatibility Matrix.

What kind of licenses are compatible with permissive licenses?

Probably all permissive licenses are compatible with each other, as they do not contain inconsistent provisions. Different license conditions must be met cumulatively. For more information the terms and conditions of permissive and copyleft licenses, check our Free Software Licenses and Compatibility Matrix.

Can copyleft licenses be compatible with other Free Software licenses?

Because copyleft licenses provide that derivative works may only be used under the same license conditions, copyleft licenses are only compatible with other Free Software licenses when one of the following conditions is met:

a) The other open-source license does not contain any license requirements that are not provided by the compatible copyleft license. This is the case with the BSD license without an advertising clause, whereas the BSD license with an advertising clause contains an information requirement that the GPL and other copyleft licenses do not provide.

b) The other open-source license contains a special compatibility or opening clause. This is the case for example in sec. 3 of the LGPL Version 2.1, that permits the use of LGPL code under the GPL. GPLv3 contains a compatibility clause for the Affero GPL and opening clauses for Apache license 2.0 and other licenses. The European Public License (EUPL) and German free software license (d-fsl) contain compatibility clauses for the GPL.

For more information the terms and conditions of permissive and copyleft licenses, check our Free Software Licenses and Compatibility Matrix.

License Compatibility and the GPL

Why are certain Free Software licenses incompatible with the GPL?

The primary goal of the GPL is the promotion and furtherance of Free Software. Accordingly, the GPL was crafted specifically to make it impossible to merge GPL covered code into proprietary derivative software works. This can be seen in the two most important of the GPL's requirements:

- Any derivative work from GPL covered code must itself be distributed under the GPL;
- No additional restrictions may be placed on the redistribution of either the original work or a derivative work.

With these conditions, the GPL succeeds in spreading the four freedoms of Free Software. Once a program is covered under the GPL, these freedoms are passed on to all other works that the code gets incorporated into, thereby making it practically impossible to use GPLed code in proprietary or closed source programs.

These same conditions however also means that the GPL is incompatible with certain other Free Software licenses. This usually happens when the other Free Software license imposes a requirement that is not present in the GPL, which makes it incompatible with the GPL's condition not to add any additional restrictions on a derivative work. Additionally, many licenses are not written with the intention to be GPL-compatible; historically, there have been much less transfers of files and snippets from one project to another.

In practice, some licenses are considered to be compatible with the GPL even though they do not explicitly contain a compatibility clause. These include the BSD 3-Clause license, and the GPL version 2.0.

Can using a GPL/AGPL-compatible license make my project easily license compatible?

Yes. The GPL is an extremely popular Free Software license. If you want your code to be freely mixed with GPLed code, then we strongly recommend that you pick a GPL-compatible license, or use the GPL itself.

What licenses are compatible with the GPL?

The Free Software Foundation maintains a list of licenses compatible with the GPL. You can find the list online here.

How are the various GNU licenses compatible with each other?

The various GNU licenses enjoy broad compatibility between each other. Do note however that you may not be able to combine code under two of these licenses when you want to use code that's *only* under an older version of a license with code that's under a newer version.

The Free Software Foundation (FSF) maintains a handy matrix for compatibility of the various GNU licenses with each other. You can refer to it here.

Summary

One challenging aspect of Free Software licensing is the necessity to maintain compatibility between various licenses in code that you have incorporated into your project. If your project has software under diverse licenses, it is important to understand the basic compatibility workflow.

CONTRIBUTIONS FROM EXTERNAL DEVELOPERS

If your software project has many external contributors (i.e. developers from outside your team who contribute to the code), this presents some issues with copyright ownership. If not properly managed, external developers may in the future claim copyright on the sections of the project that they contributed to. This issue has come up various times in the past in Free Software projects.

Accordingly, you may wish to consider whether steps should be taken to ensure that ownership of the entirety of the code belongs to you. Typically, it would be prudent to refrain from accepting external contributions into your repository without a clear contribution policy, whre you can set specific instructions on how contributions are accepted. In this part you will learnb more about ways to accept external contributions in a transparent, organized and safe way.

External Contribution Policy

What is an external contribution policy?

An external contribution policy is a set of rules that govern the management of third party Free Software within a project. More precisely, it states how external contributions are handled and how they are audited and distributed.

If you want to build a community around the project, and accept contributions from developers outside your team, a simple and clear external contribution policy can help your project to maximize the impact and benefit of using Free Software, while at the same time mitigating technical and legal risks.

For more information, you can refer to our Guidelines for External Contribution Policy.

What should a contribution policy contain?

A simple, clear and lightweight policy can be easily implemented by creating a file where you indicate the rules of contribution and another one with the legal mechanism to enforce this rules.

Since an external contribution policy is not a strictly legal document, it makes sense to have a a legal instrument that you can point contributors to in order to protect your project from code contributions that may infringe third party copyright or patents. Depending on the size and compexity of the contributors network, there are different legal instruments you can use to manage external contributions, namely DCOs, CLAs, and CAAs.

For more information, you can refer to our Guidelines for External Contribution Policy.

Where I can find a template for a contribution policy?

You can find more information on how to create an external contribution policy for your project in our Guidelines for External Contribution Policy.

Contributor Agreements

What are contributor agreements?

Contributor agreements are agreements between a Free Software project and contributors to that project that set out what the project can do with the copyright of the contributions made; whether it be the code, translations, documentation, artwork, etc.

The purpose of such agreements is to make the terms under which contributions are made explicit, and thereby protect the project, the users of the project's code or content, and often also the contributors themselves. They provide confidence that the guardian of a project's output has the necessary rights over all contributions to allow for distribution of the product under either any license or any license that is compliant with specific principles.

The goal of such projects would then be to :
a) allow the relicensing of a project under another Free License;
b) enable the enforcing of copyright in the case of violations; and/or
c) establish a dual licensing model.

What are the different types of contributor agreements available?

There are three major categories of contributor agreements: Copyright License Agreements (CLAs), Copyright Assignment Agreements (CAAs), and Fiduciary License Agreements (FLAs).

Whereas the CLA require an irrevocable license to allow the project owner to use the contribution, the CAA requires assignment and therefore a transfer of copyright to the project owner. A FLA on its turn does not requires copyright assignment, but involves only contribution handling and administration.

You can find templates for contributor-friendly, multi-purpose contributor agreements, including for CLAs and FLAs on the following website: https://contributoragreements.org.

Contributor License Agreements (CLAs)

What are Contributor License Agreements (CLAs)?

A Contributor License Agreement (CLA) is a legal document that defines the terms under which intellectual property, including software code, is contributed to a larger project. In general, the goal of a CLA is to grant sufficient rights to a Free Software project, to allow the project to release a software contribution under the project's Free Software license(s).

Some CLAs may state that once the contributor has provided a project with a contribution, they cannot try to withhold permission from the project for use of the contribution at a later date. For a simple case, such a CLA may require that a contributor to the project assign ownership of the copyright in the contribution over to the general project.

Nevertheless, CLAs are not standardized, and contributions to different Free Software projects may be subject to different CLAs, or even none at all.

Nowadays, CLAs can come in an electronic form that a developer fills out and sends in to the project, or even a web-based checkbox that the developer checks before completing their first contribution to the project.

You can find templates for contributor-friendly, multi-purpose contributor agreements, including CLAs and FLAs on the following website: https://contributoragreements.org.

What kind of terms should be included in a CLA?

The purpose of a CLA is to ensure that the project has the necessary ownership or grants of rights over all contributions to allow them to distribute under the chosen license. Accordingly, the contributor needs to at the very least grant the rights that will be granted to the project's owner in the license to be used for the distribution of the overall project.

When granting rights it is common to grant a very broad range of rights. This is in order to avoid the need to return to the contributor for authorization to take a desired action with their contribution, such as releasing under a different license.

What are the benefits of using a CLA?

By expressly describing the rights and obligations of contributors, the software project itself, and/or its maintainers, a CLA can protect the project's participants from disputes regarding licensing or ownership of software contributions.

A CLA can also provide legal assurances for the software project. In this case, the software maintainer can rely on the CLA that a contributor:

a) has the right to make their contribution to the project; and

b) is not prevented from making such a contribution because of any intellectual property rights of an employer.

CLAs can also include other provisions that provide certainty over the conditions of the licensing of the project in general, which is beneficial for the software project over the long term. Such provisions can address licensing issues, such as:

a) clarifying who will be responsible for enforcing the project's license in the event of copyright infringement;
b) providing permissions for the project to change the license adopted in the future, without having to seek authorization from each of its contributors before making such a change;
c) providing permissions for the project to distribute the contribution simultaneously under separate licenses, for example in a dual licensing scheme with a proprietary license.

Additionally, CLAs can serve as a formal mechanism for the software project to keep track of its contributors and contributions, if each contributor provides relevant identifying information in the CLA.

What are the disadvantages of using a CLA?

Nevertheless, there may be reasons why a project might not wish to use a CLA. For one, CLAs have the potential to discourage contributions. Having a contribution to a project be accompanied by a legal contract can be intimidating to software developers, especially when they simply wish to contribute minor bug fixes or other refinements.

The CLA can therefore be seen by some to be a barrier to entry that discourages some developers who would otherwise contribute to the project. This is especially so the more complicated the terms of a CLA gets; if a contributor does not fully understand the terms of the agreement, and feel that they need to sign it to contribute, then they may perceive the CLA to be coercive. Additionally, contributors may wish to remain anonymous, which in all likelihood is not possible under a CLA. Because of these reasons, a CLA may end up discouraging some developers from making contributions.

Cataloging and maintaining a database of CLAs received for each contribution can also require some effort. Especially for projects that have a large number of contributors, this work can be a non-trivial task.

What kind of contributions require a CLA?

As some contributions may be very small, for example, simple bug fixes or spelling corrections, you may be tempted to bypass requirements for CLAs for small contributions. For small contributions, the risk of any loss suffered from any potential legal complications can be small.

Conversely, the larger the contribution, potential losses from legal complications not covered by a CLA can be much higher.

While it is ultimately up to the project owners to decide which contributions they choose to accept with and without a CLA, we recommend using a CLA in all cases to avoid interpretation problems of what is copyrightable and what is not in your software project. It is very important to record the assignment of copyright or grant of rights for each contribution and from each contributor.

Contributor Assignment Agreements (CAAs)

What are Contributor Assignment Agreements (CAAs)?

A Contributor Assignment Agreement (CAA) is a legal document that effectively transfers ownership of the copyright over contributed material from the contributor to the project. This differs from CLAs, which give the licensee a license to use the copyrighted material, while the contributor still retains copyright ownership.

CAAs can create complex legal questions in many jurisdictions, and accordingly we do not recommend that you adopt them for your software project.

Fiduciary License Agreements (FLAs)

What are Fiduciary License Agreements (FLAs)?

A Fiduciary License Agreement (FLA) is a legal document that effectively concentrates all deciding power over the software project within one entity. On the one hand, this can prevent the fragmentation of rights, while on the other preventing that single entity from abusing its power.

This process only serves for the transfer of economic rights. So-called moral rights (e.g. the right of authors to be identified as the author) remain with the original author and are inalienable.

You can find templates for contributor-friendly, multi-purpose contributor agreements, including CLAs and FLAs on the following website: https://contributoragreements.org.

How does an FLA function?

An FLA is an agreement between two parties: the Contributor to the project on the one hand, and an entity who would be the new holder of rights on the other (the Trustee). As all rights will be concentrated in the Trustee, the fragmentation of rights among many contributors can be

prevented. It ensures that copyright-relevant modifications can be done easier, without the need to pin down all existing contributors.

The Contributor in an FLA does not lost all of their rights, just the exclusivity in them. While the Contributor transfers their exclusive rights to the Trustee, the Trustee reciprocally grants the Contributor a non-exclusive worldwide, royalty-free, perpetual and irrevocable license to the same extent as it was originally transferred from the Contributor.

What happens if the Trustee misuses the rights?

In the event that the Trustee misuses the rights granted to them by Contributors, for example by relicensing the Free Software as a proprietary one, an FLA can offer a special clause to protect against such a situation. According to this provision, if the Trustee acts against the principles of Free Software, all granted rights and licenses return to their original owners. This effectively prevents the Trustee from continuing any activity contrary to the principles of Free Software.

Who can be a Trustee?

Anyone can be a trustee, whether a natural person, or an entity such as a company or organization. However, before becoming a Trustee, it is always important to assess all the risks connected with becoming one. Being a Trustee is a responsibility that should not be underestimated.

Developer Certificates of Origin (DCOs)

What are Developer Certificates of Origin (DCOs)?

Developer Certificates of Origin (DCOs) is a legal mechanism that is often used as an alternative to CLAs. DCOs are an affirmation that the contributor intends to contribute the contributed code under the project's license, and that the contributor has the right to do so. It was introduced in 2004 by the Linux Foundation to enhance the submission process for software contributions used in the Linux kernel.

In using a DCO, external developers certify that they adhere to these requirements by signing off on their commits. Typically, this would mean that the external developer offers their changes under the same license as the software project they are contributing to, that they had the right to do that, and that they did not contribute someone else's work to the software project other than their own.

If you are concerned about developers not contributing to your project because they might be put off by the legal complexity of a CLA, we recommend that you nevertheless adopt a DCO. DCOs can give developers flexibility and portability over their contributions by allowing developers to retain ownership over their contributions.

You can find a template text for a Developer Certificate of Origin here: https://developercertificate.org/.

What should I use to handle external contributors for my software project?

It depends.

DCOs set the incoming license of the external contribution to be the same as the outgoing license of your software project.

On the other hand, using a CLA reserves the right for you to decide whether to relicense the entire software project (and accordingly all past contributions to it) under a different license in the future, if the current license used for your project is found to be inadequate down the road. This can be a benefit as you would not be required to contact everyone who has contributed to the project to obtain their permission to do so. However, this presents a social risk, that the project takes a direction that contributors find unacceptable, and find themselves unable to act upon it. Using CLAs for your external contributors also add an additional layer of bureaucracy for your project, and would require resources to draft and record, but they do provide more concrete protections for your project. Conversely, DCOs reduce the barriers for external developers to contribute, while still requiring them to certify that they are submitting their own work.

CAAs are a more heavy-handed manner of ensuring that the software project always has the rights to use the code that has been contributed. We generally do not recommend that you adopt CAAs for your project.

FLAs are similar to CLA but do not involve copyright assignment, letting the contributors to retain their rights. Instead the holder of rights (trustee) grants a broad license back to the contributor.

Nevertheless, you should consider the needs of your specific software project and the community that it serves, and decide accordingly. Apart from DCOs, CAAs and FLAs can be legally complex, therefore we recommend you to contact FSFE team for NGI0 or a lawyer in your jurisdiction.

Where can I find a suitable contributor agreement for the purposes of my software project?

You can find templates for contributor-friendly, multi-purpose contributor agreements including CLAs and FLAs on the following website: https://contributoragreements.org.

In particular, you can use the Agreement Chooser here to select a template agreement: http://contributoragreements.org/agreement-chooser.html.

Additionally, you can find a template text for a Developer Certificate of Origin here: https://developercertificate.org/.

Summary

How does the project get the legal right to redistribute some contributor's code changes? How does the project know the contributor won't sue later for copyright infringement? In this section you will have a handy overview on how the contributor *gives* the project that right formally, by signing an agreement saying that the project can redistribute the contributed code.

Free Software and Software Patents

Software patents are a highly controversial topic, with negative consequences for Free Software. In general terms, there are two topics which patents can concern your project: when your project owns patents or you produce software under patent owned by a third party. Please note that if your project has software patents involved, please contact us directly.

Patents

What are patents?

A patent is a type of intellectual property separate from copyright and trademarks. It grants an exclusive property right for an invention that enables the patent holder to exclude other people from practicing the invention or idea claimed in the patent.

How are patents different from copyright ?

While copyright protects the *expression* of ideas, patents work to protect the *ideas* themselves, not just the expressions of them. The main effect of a patent is therefore to give patent holders the right to challenge any use of the invention or the idea by another person.

Whereas a piece of code is regarded as an expression that is protected by copyright, an invention can be implemented by different expressions, and therefore in different copyrightable programs.

Software Patents

What are software patents?

A software patent is a patent on an invention which is implemented in a piece of software or in a combination of software and hardware.

An example of a software patent would be the patent over Amazon One-Click, a software that lets internet shoppers purchase something on their web platform with a single click, if a user has their payment, billing, and shipping information saved.

Patent law varies fundamentally among countries. In Europe, the patent system is run by the European Patent Organisation (EPO), a supranational organ governed by the European Patent Convention of 1970. Although many countries place limits on the patenting of inventions involving software, including the European Patent Convention, the EPO has already granted thousand patents related to computer programs.

Yet, there is no consensus about the definition of a software patent. What the European legislation agrees is, in contrast to copyright (which is exercised over the code itself), software patents are exercised over the **functionality** in which the software is intended to work. It is independent of the code. The EPO argues that as soon as a computer program has a "technical effect" – making a hard drive spin, lighting up pixels on a screen – it is a physical machine, and therefore patentable.

What does it mean that the "functionality" of the software can protected by patents?

Under copyright law applicable to software, if part of your code infringes on someone else's copyright, you can solve this issue by rewriting the offending section of code, while continuing to ensure that the underlying function that the code aims to achieve remains the same.

However, in the case of a software patent, it does not matter how the code is written, or what programming language is used, as the patent acts as a blanket restriction against anyone but the patent holder from implementing a certain idea. Accordingly, once a patent holder accuses a software project of infringing a patent, the project must either stop implementing the offending feature, or expose the project and its users to legal risk.

This can have a negative impact on Free Software projects. For example[12]:

- Software patents add legal risks, and therefore costs, to software development.
- Software patents specifically inhibit the development of useful software by blocking compatibility and interoperability.
- Patents are incompatible with software because software is so complex – too many ideas are used for it to be practical for a developer to check each individual idea against existing patents, to prevent infringement.

One of the major downsides of the patent system is that enforcing a patent does not require the patent owner to engage in any business he/she is trying to protect. This is why patent "trolls" can exist – companies that do not engage in any business except suing other for patent infringement.

Nevertheless, Free Software projects has a tendency to be less vulnerable to patent claims. Among the factors is fact that code used in Free Software is freely available to review, avoiding legal discovery procedures, as well as due to the cooperation on engineering among Free Software community to find solutions around patent claims. Besides, Free Software licenses contain liability limitation for the developer, what can restrict in some cases the liability for patent infringement[13].

12 Read more: http://en.swpat.org/wiki/Software_patents_wiki:_home_page
13 Read more: Meeker, Heather. *Open Source for Business.* 2.ed. Fleming, 2017. p. 153-178.

My project owns software patents. How does a Free Software license affects it?

Free Software licenses are primarily copyright licenses, but many of them have expressed and implied patent licensing built into them. So, if you foresee handling software patents in your project, here are some aspects that you should be aware of.

While traditional patent licenses usually seek to slice the patent rights granted into narrow pieces, Free Software licenses are purposely broad and vague, mainly because if someone distributes Free Software and holds software patents, a absurd situation may arise where parties may use, study improve and share the software, but simultaneously be prohibited from using the software by the software patent.

Therefore, in general terms, it is possible with Free Software licenses to:

- **Grant patent rights**:
  A Free Software license may require that distributors of a software give recipients a licence to use any necessary patents. Some licenses contain expressed grant but other grant patent rights by implication. Even in the absence of an express patent grant, all Free Software licenses grant patent rights in some extension. An Free Software patent license grant has only one field limitation, which is that the right is granted only in connection with the exercise of the copyright granted for the software. Any other field limitation, such as territory, commercial or technology fields is against Free Software definition and therefore not included.
- **Retaliate in case of patent aggression**:
  A Free Software license may have the effect of making patent aggression less attractive by revoking patent rights that any aggressor received through the license. A **defensive termination** revokes the patent grants in case if someone exercising the license brings a claim accusing the licensor of patent infringement.

How do I know if my software is already patented?

Patent is an monopoly on using an idea. Software patents are exercised over the functionality in which the software is intended to work. It is independent of the code. Therefore it is very hard to determine if a software that you are developing has patented parts on it. That's why the GPL v.3 states in its preamble that every program is threatened constantly by software patents.

A detailed search for patented software can be costly and involve lawyers and patent experts. It is quite common for developers to do a patent search and find nothing even when there are things that could be found by a professional searcher.

Nevertheless, there are also initiatives to minimize patent aggression towards the Free Software community, for instance the Open Invention Network, which enforces a license agreement among organisations in support of patent non-aggression and for free access to OIN's patents.

Does my Free Software license provide patent grants and/or retaliation clause?

The following table[14] on the next page lists the patent grants terms and defensive termination provisions of the recommended Free Software licenses. Please note that some license text excerpts were edited for better reading.

License/ SPDX code	Grant Patent Rights	Termination Trigger	Rights Terminated
GNU GPLv3 GPL-3.0-only	**Section 11** **Express grant** *Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.*	**Section 10** *You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may (…) initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim in infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.*	**Section 8** *All rights can be terminated, including any patent licenses granted.*
European Union Public License v1.2 EUPL-1.2	**Clause 2** **Express grant** *The Licensor grants to the Licensee royalty-free, non-exclusive usage rights to any patents held by the Licensor, to the extent necessary to make use of the rights granted on the Work under this Licence.*	**Clause 5** *The Licensee (becoming Licensor) cannot offer or impose any additional terms or conditions on the Work or Derivative Work that alter or restrict the terms of the Licence.*	**Clause 12** *The Licence and the rights granted will terminate automatically upon any breach by the Licensee of the terms of the Licence.*
GNU AGPLv3 AGPL-3.0-only	**Section 11** **Express grant** *Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.*	**Section 10** *You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may (…) initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim in infringed by making, using, selling, offering for sale, or importing the Program or any portion of it*	**Section 8** *All rights can be terminated, including any patent licenses granted.*
GNU LGPLv3 LGPL-3.0-only **Note:** This license incorporates the terms and provisions of **GNU GPLv3.**	**Section 11 (GNU GPLv3)** **Express grant** *Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.*	**Section 10 (GNU GPLv3)** *You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may (…) initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim in infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.*	**Section 8 (GNU GPLv3)** *All rights can be terminated, including any patent licenses granted.*
Mozilla Public License 2.0 MPL-2.0	**Clause 2.1** **Express grant** *Each Contributor hereby grants You a world-wide, royalty-free, non-exclusive license: under Patent Claims of such Contributor to make, use, sell, offer for sale, have made, import, and otherwise transfer either its Contributions or its Contributor Version.*	**Clause 5.2** *If You initiate litigation against any entity by asserting a patent infringement claim (excluding declaratory judgment actions, counter-claims, and cross-claims) alleging that a Contributor Version directly or indirectly infringes any patent.*	**Clause 5.2** *The rights granted to You by any and all Contributors for the Covered Software under Section 2.1 of this License shall terminate.*
Apache License 2.0 Apache-2.0	**Clause 3** **Express grant** *Each Contributor grants to You a perpetual, worldwide, non-exclusive, no-charge, royalty-free,*	**Clause 3** *If You institute patent litigation against any entity (including a cross-claim or counterclaim in a law suit) alleging that the*	**Clause 3** *Any patent licenses granted to You under this Licese for that*

14 Table built according to Meeker, Heather. *Open Source for Business.* 2.ed. Fleming, 2017. p. 167-169.

	irrevocable patent license to make, have made, use, offer to sell, sell, import, and otherwise transfer the Work,where such license applies only to those patent claims licensable by such Contributor that are necessarily infringed by their Contribution(s) (...)	*Work ir a Contribution incorporated within the Work constitutes direct or contributory patent infringement.*	*Work shal terminate as of the date such litigation is filed.*
MIT License MIT **Note:** The MIT license does not include an express patent license. However, there are opinions that supports the interpretation of implied patent grant provision.[15]	**Implied grant** *The permission is here granted to deal in the Software without restriction, and to permit persons to whom the Software is furnished to do so (…)*	**General termination conditions** No copyright notice and permission notice included in all copies or substantial portions of the Software.	**General termination of rights** The grants are subject to the license's conditions.

Summary

Although highly controversial, software patents are a reality that many projects should face. In this section you will learn how software patents are covered by the Free Software licensing terms.

15 Read more: Peterson, Scott. *Why so little love for the patent grant in the MIT License?,* 2017. https://opensource.com/article/18/3/patent-grant-mit-license