Free and Open Source Licenses, Software Development, and Distribution

by Stephen Walli | VP, Open Source Development Strategy

Programmers have been sharing computer programs and source code since we had computers. In the early days it was often done through professional and user organizations such as DECUS, SHARE, and USENIX. The licenses through which such sharing happened were as varied as the end user license agreements (EULA) of proprietary software vendors today, and all such licenses rely on strong intellectual property laws and copyright law.

This sharing of software has reached new heights over the past couple of decades enabled through the ease of sharing across the Internet. The concepts of "free" and "open source" software have became mainstream and licensing is the avenue through which the rules of this particular form of sharing software are laid down.

Understanding free and open source (FOSS) licenses is not actually that difficult. A little history and a few pointers can clarify some of the confusion to enable organizations to make best use of FOSS in their own business contexts. With this understanding we will take a look at enterprise considerations around FOSS use, whether as an enterprise that wants to use FOSS in parts of its business infrastructure or a vendor looking for a competitive edge and a new value proposition for its customers.

The Licenses

There are primarily three license types or families that have arisen historically:

- Academic licenses (MIT Athena, Berkeley, and Apache)
- Free software licenses (General Public License and the LGPL)
- Mozilla-style licenses (Mozilla, and the IBM licenses)

We will note a few other interesting licenses along the way, but even these derive from the basic models laid down in these three groups.

The Academic Licenses (Berkeley, MIT, Apache)

During the mid-1980s period, the Computer Science Research Group at the University of California, Berkeley was doing a lot of research work on early UNIX systems, and acted as a hub for the collaborative research community. The regents of the university developed a simple license for their work to encourage new research and adoption of the software. The Berkeley license essentially:

- Enables the software user to do anything with the software, including extending and selling it.
- Does not require any derived software be licensed under the same license or that the changes be published. This enables "closed" or proprietary products to safely include such licensed software.
- Requires that attribution be given for the work, and copyrights maintained.
- Disclaims any warranties (express or otherwise) just as proprietary EULA do.

This license style has been referred to as Berkeley-style licensing. This was also the basic model for the MIT Project Athena license (used for the X11 windowing technology, including all the contributions from Hewlett-Packard and Digital Equipment Corporation).

As we will see, Berkeley-style licensing supports a reciprocity belief counter to that espoused by the Free Software Foundation (FSF) – that the software would definitely be freely distributable, but the reciprocity requirement should be encouraged in the community and not commanded in the license.

The Berkeley-style license was also the model used in the early Apache community in 1995. The original Apache web server was created out of work developed at the National Center for Supercomputing Applications, University of Illinois, and the license reflected the research base and collaborative development in that community.

In 2004 the Apache 2.0 license was released. It is a complete rewrite to account for current concerns of software contributions and patents, and is a richer and more complex license in its legal structure, but it remains true to the principles of its history.

Free Software Licenses (GPL, LGPL)

In 1985, Richard Stallman created the Free Software Foundation and his definition of software freedom, where a program's source code was always available and a user could always fix and extend the software without restriction. The General Public License (GPL) laid down this particular sharing foundation.

- If the user distributes the changed software they can only do so by sharing their changes the same way through the same license. This is the reciprocity requirement of the free software definition. This is a primary difference from the academic class of licenses that permit derivatives to be re-licensed under other (possibly closed, proprietary) terms.
- If you used any of the GPL-licensed source code in your own programs, and distribute those programs, the entire newly derived program including your own source code becomes subject to the GPL. This is where the concept of a virus is attached to the GPL.
- The GPL disclaims any warranties (express or otherwise) just as proprietary EULA do.

It is important to note a couple of things here:

- The reciprocity requirements are triggered on distribution of the software, not on using it.
- There is nothing that has forced you to expose the source code to your application. The license contains its own redress. You can always withdraw the software distribution. (If you were a commercial software organization, this might still prove onerous, and so one does need to pay attention when working with GPL software that will be distributed.)

The Lesser GPL (LGPL) was developed later to account for software libraries. Many that would share their software subroutine libraries under the GPL didn't necessarily want to force the recipient to have to share anything other than their changes to the library. The way the GPL was written would unfortunately force the entire software (libraries and the program using the libraries) to come under the GPL. The LGPL enabled a library to be licensed which did not require the entire application to be licensed under the same license (and so enabling it to remain closed), while still requiring changes to the library itself to be published under the LGPL if distributed.

Many of the most important FOSS programs of the past 20 years are licensed under the GPL, including the Linux operating system, the GCC compiler suite, the MySQL database engine, and JBOSS application server. Many vendors (including software vendors) use and develop software licensed under the GPL.

All through much of the rest of the 1980s and 1990s everyone followed one of these two models with simple variations around such clauses as jurisdiction.

The Mozilla License

Before we cover the Mozilla license, a small detour is in order. When the Perl language hit the scene, Larry Wall created the Artistic License. The Artistic license was intended to maintain the open aspect of the Artistic licensed code, while enabling innovation around the core project to be licensed as appropriate. It tried to find a balance between the hard line sharing required by the GPL and the complete freedom of the academic licenses. It is a popular license, though some consider it legally ambiguous in places.

In the late 1990s, Netscape published the source code to their browser and began to build a community of developers around it. This project was called the Mozilla project, and the license created was the Mozilla Public License (MPL). This is one of the first licenses created by a corporation, and that heritage shows through in its legal structure and depth compared to FOSS licenses prior to that point. It had similar goals to the Artistic License. Essentially, the MPL:

- Requires derivatives of the MPL work that are the original work plus contributions to be licensed under the MPL, thus creating the reciprocity of the GPL for the core project.
- Enables MPL licensed works to be combined with other software and re-licensed into a “Larger Work.” This enables the development of possibly closed proprietary software similar to the academic licenses.
- Discusses patent rights relevant to the licensed work.
- Disclaims any warranties (express or otherwise) just as proprietary EULA do.

There has been a proliferation of open source software licenses based on the Mozilla license, because other companies wishing to develop collaborative software communities as a business tool invariably want to change the jurisdiction clause and define language around what patent concerns they may or may not have. The language of the Mozilla Public License is very Mozilla project centric.

One can see a certain lineage to the Mozilla license in the development of IBM licenses, from the original IBM Public License through the Common Public License to the newest Eclipse Public License that are used around the Eclipse project.

Enterprise Considerations

For the most part, enterprises using free and open source software should have few concerns about licensing for the following key reasons:

- All licenses essentially allow the software to be run (binary form) without restriction
- Under all licenses the source code can be modified without restriction if the resulting software is being used internally
- The GPL and Mozilla family of licenses place requirements for re-licensing and publication on the user only if they distribute the software. This would only have implications on an enterprise if they plan to distribute the software to their customers. (Software development and distribution concerns are discussed in "Free and Open Source Licenses, Software Development, and Distribution".)

If you're buying packaged free or open source software or a system that contains such software (e.g. Red Hat Advanced Server, or HP/UX), then the Red Hat or HP EULA is the primary concern, and all other third party license concerns are left to the vendor.

When using open source packages, such as the MySQL database engine, JBOSS application server, or any of the Java frameworks that are FOSS licensed, the license enables free deployment and use and there are no concerns within an enterprise – the enterprise isn't developing software derivatives that they distribute.

Indeed, the ability to freely copy open source software and deploy as much as is needed within an enterprise means the historical (and sometimes litigious) problem of counting users or processors goes away, along with the auditing costs involved. This ability to freely deploy also frees up the architecture of solutions to problems. For example, there was a time when you designed the solution architecture around reducing the number of very expensive licenses one required for application server middleware, and database access. With the ability to deploy as many application servers as is required and distribute the database across systems equally freely because of a lack of per system license fees, the solution can be designed and built to real requirements. The solution can grow more organically to meet the needs of the enterprise at marginal additional costs. The issues then fall back to concerns about support and maintenance.

Getting More Information

The following web sites and books are excellent sources of additional information on free and open source software licensing.

Web sites:

- The Open Source Definition (http://www.opensource.org/docs/definition.php)
- The Free Software Foundation definition of free software (http://www.fsf.org/licensing/essays/free-sw.html)
- Open Source Initiative approved licenses referenced in this document can all be found at the following web site: http://www.opensource.org/licenses/

Books:

- Lawrence Rosen, Open Source Licensing, Prentice Hall PTR, Upper Saddle River, NJ, 2004 (ISBN 0-13-148787-6)
- Andrew M. St. Laurent, Open Source and Free Software Licensing, O'Reilly Media Inc., Sebastopol, CA, 2004, (ISBN 0-596-00581-4)

CONTACT
Brian Otis
VP, Sales and Partnerships
email: botis@optaros.com
phone: (617) 227-1855 x110