Contract n 14400.2005.004-2007.717 CS 6
Framework contract ENTR/05/066/IDABC/OSOR
IDABC Open Source Observatory and Repository (OSOR)

Guidelines on publishing and sharing existing software as Open Source

Specific Contract No 6 (SC6)
of 20 December 2007

X- DIS /OSS Phase 3

Version 2.0

Authors: Rüdiger Glott (MERIT)
Patrice-Emmanuel Schmitz (UNISYS)
Editor: Emidio Stani (UNISYS)

May 6, 2009

Table of Contents

1 Purpose of these guidelines....4
2 Reasons for public sector institutions to engage in Open Source Software 4
3 Guidelines 6
  3.1 Make sure you are the copyright owner of the software....6
  3.2 Be aware of “marketing” issues....7
  3.3 Decide how you want to distribute your Open Source Software (Licensing issues)....12
    3.3.1 General licensing issues....12
    3.3.2 Dual licensing 14
    3.3.3 Practical issues....14
    3.3.4 Tracking intellectual property....15
4 Technical infrastructure 16
5 Releasing....17
  5.1 Recommendations....17
  5.2 Source Code....19
  5.3 Documentation....20
  5.4 Packaging....20
  5.5 Installer 21
6 Good practices for using the OSOR forge 22
7 Making use of the OSOR community “OSS and Statistics”....23
8 Helpful software....24

1 Purpose of these guidelines

This document gives the guidelines for packaging and publishing Eurostat software as Open Source. Its intended audience are project officers and contractors. The purpose of this document is to contribute to advancing the existing guidelines for writing and publishing software as Open Source. Fogel[1] points out that Open Source Software distribution combines two fundamental tasks: The software needs to acquire users, and it needs to acquire developers. The following paragraphs explore aspects of the open source software distribution process from the perspective of the copyright owner because owning the copyright is an indispensable condition for distributing the software.

These guidelines will cover the following aspects:

a) Understanding why Open Source Software is an opportunity for the public sector
b) Understanding copyright aspects related to the software and its components;
c) Examination of licences and their aspects, including the European Union Public Licence (EUPL) and its compatibility with other licences;
d) Recommendation of the appropriate distribution licence (single or dual licensing);
e) "Practical guidelines" (e.g. mentions to insert in the source code, website or repository for accessing code and licensing information).
f) Prepare the work and define objectives;
g) Identify components or solutions that could be distributed;
h) Organise a distribution strategy;
i) Identify the distribution licence;
j) Organise a distribution team (leader, sponsors, contributors, other stakeholders);
k) Motivate the internal community;
l) Enlarge community and motivate external contributors;
m) Pool efforts with other stakeholders sharing the same needs.

These steps and corresponding efforts are not equally important, depending on the ambition of the project. It is also obvious that some of these steps (such as pooling efforts) may obviously be present in "non Open Source" approaches as well.

2 Reasons for public sector institutions to engage in Open Source Software

The information economy is now in the range of 10% of GDP in most developed countries, and Public Sector participates to a very substantial part of it (20%)[2]. Governments are trying to optimise their methods, to build more efficient interaction with citizens and enterprises (through e-Government) and to optimise the instruments that are needed by decision makers to adopt the right policies (through developing reliable tools providing a correct picture of the reality and facilitating the forecast of the future).

[1] Karl Fogel (2007): Producing Open Source Software - How to Run a Successful Free Software Project. Available online at http://producingoss.com/en/index.html.

[2] For detailed report, see the November 2006 study on the "Economic Impact of Open Source Software on innovation and the competitiveness of the ICT sector in EU" – By Rishab Aiyer Ghosh – UNU-Merit

If you or your administration participates to this investment, you must be committed to obtain the best value from taxpayer's money, and to ensure that investments will not become obsolete within a short period.

Software goods are not like material goods: they can be duplicated and distributed without any additional costs. Even more important, their value may increase with wide use and distribution. At the contrary of a diamond or of a famous painter's work (which have more value if they are unique), software will gain value if it is used by a wide community of users at a point to become "a recognised standard".
In the hands of proprietary software industry, such situation may become a monopoly. In the hands of public sector that have no commercial objective, such promotion of their work may provide more chances that investments are sustainable, will be developed and maintained because responding to a larger need. This could happen if:

a) The software is used, corrected and improved by the widest possible community;
b) Improvements are returned to everyone (including the original donator) and are not exclusively appropriated.
c) Other public administrations (in other countries with similar needs or obligations) do the same, accelerating comparison, the selection of best practices and innovation.

Open Source Software provides a software distribution model corresponding to the above conditions. Open source software is software where the author (the 'licensor') gives a number of fundamental freedoms to the user (the 'licensee') via a license agreement. These freedoms include the possibility to study how the programme works, to adapt the code according to specific needs, to improve the programme, to run it for any purpose on any number of machines and to redistribute copies to other users[3]. It is necessary to note that just allowing access to the source code of a software system, or allowing its free download does not make it open source. Precisions concerning the term "Open Source" and the related freedoms are published by a non-profit organisation, the OSI[4]. The term "Free software" is a quasi synonym promoted by another organisation, the FSF[5] and the acronyms FOSS or FLOSS are frequently used to combine both terminologies[6].

The open source software development and business models differ from the closed source or proprietary models that are generally applied by the large software industry vendors. Differences include the way the software is developed, bundled or packaged, the roles played by participants and the way to obtain support if you position yourself as a user. Both models, Open Source and proprietary, share common issues such as security, quality and as far as applicable warranty and liability.
From the public sector point of view, the Open Source model represents several advantages:
a) Using software at reduced entry cost (as it is the case for any other Open Source user)
b) Promoting and distributing its own software production, which produces tangible benefits: the more a software is used and becomes a "standard" the greatest is its value, its number of users as the number of technical experts (developers, business specialists) that will rely on it

[3] IDABC – Open Source introduction: http://ec.europa.eu/idabc/en/chapter/468
[4] OSI – Open Source Initiative: http://www.opensource.org/docs/definition.php
[5] FSF – Free Software Foundation: http://www.fsf.org/
[6] FOSS = "Free & Open Source Software", FLOSS = "Free, Libre & Open Source Software" to express clearly that "Free" means "freedom " and not that the related software is always given for free. FLOSS is becoming the preferred acronym used in specialised European studies.

and improve its quality.

c) For public sector, Open Source licensing facilitates such sharing by protecting the public administration copyright and avoiding (depending on the licence) that the software could be appropriated by a third party. It fits also with the public service philosophy, which is generally not focused on constructing commercial proprietary business by selling software that was already funded by taxpayer's money.

3 Guidelines

3.1 Make sure you are the copyright owner of the software

Computer software is protected by copyright law. Copyright law gives the owner of a work certain rights over it, and makes it illegal for others to use the work as if it were his/hers. Owning the copyright in a piece of work, whether literary or programmatic, means that the author decides who can copy it, adapt it and distribute it. By default, only the owner can do these things. Anyone who copies, changes or distributes someone else's work without permission can become subject of legal action. Copyright comes into being automatically as soon as a work is 'fixed', meaning as soon as the work is recorded in some way. There is thus no need for gaining copyright, to register the work or to mark it s copyrighted, with a © symbol. Writing software may also well result in more than one piece of property: a program's source code is property, as can be the preparatory design material for it, its general organisation and its user interface.

When someone writes software, he/she creates property. By default, this property will be owned by somebody. As a piece of Open Source software is developed often by many different people and groups, its ownership becomes more and more complex. Issues relating to collaboration and ownership apply also to unplanned collaborations over a period. It is conceivable for every contributor to own the copyright to their contribution.

In the case of the public sector, ownership of Open Source Software can be generated as follows:

a) If the writer is an employee or an official of a public administration, it is likely that this administration will own the software created.

b) If the developer is working for him/herself, or working in his/her free time on matters unrelated to the work he/she is supposed to deliver, it is likely that he/she will own it.

c) On the contrary, if the developer is working for the administration as a service provider under a contract agreement, the owner of the resulting intellectual property should be defined in the contract. If the contract does not define who will own the property, then it is likely that the contractor who ordered the work will own it. Ideally, it should be agreed on who will own what before the works begins.

In several cases, the situation is not quite as clear: if parts of the existing development were done by contractors, it must be clarified if the development contract confers full property rights to the public authority, with the right to redistribute it without any limitation. Such limitations may concern situations where the software is only licensed, or could be redistributed:

- only into the same administration
- to a limited number of users
- to a class of users (officials, civil servants)
- for a specific country or geographic area
- for a limited purpose (public administration, non-commercial)
- etc.

If any of such limitation exists, there is no full ownership on the related components, meaning that the component should be:

- changed (adopted from another Open Source project with similar functionalities)
- adapted (from an existing Open Source project with close functionalities)
- rewritten (replacing proprietary code by new, original code).

Quite often, development contracts contain a declaration of "transferring ownership" or "exclusive ownership" to the public authority, meaning that the contractor transfers all his rights to the public authority and will not re-sell the same solution. Such a contract should be reinforced by authorising public disclosure of the code (and granting the public authority an indemnity against actions in case of copyright or patent infringement) and authorising public licensing of the code by the public authority (and granting the exact model of licence that could be used then, depending on the possible licences of the components embedded in the software).

3.2 Be aware of "marketing" issues

The explicit purpose of software distribution is to spread the software and to increase the user and developer base. In order to attract attention of others and to ease the adoption and further advancement and distribution of your software, compliance with a few basic recommendations should be considered:[7]

- Choose a Good Name
  - A good name should give some idea what the project does, is easy to remember, is not the same as some other project's name, and does not infringe on any trademarks.
  - If possible, the name should be available as a domain name in the .com, .net, and .org top-level domains. Fogel recommends to pick ".org" in order to advertise as the official home site for the project; the other two should forward there and are simply to prevent third parties from creating identity confusion around the project's name.
- Have a Clear Mission Statement
  - A mission statement serves the purpose to provide others with a quick description of the project and the software. It should be so short and clear that readers can decide within 30 seconds whether or not they're interested in learning more. The mission statement should be prominently placed on the front page, preferably right under the project's name.[8]

[7] This section follows the recommendations given by Karl Fogel (2007): Producing Open Source Software - How to Run a Successful Free Software Project. Available online at http://producingoss.com/en/index.html.

[8] Fogel (2007) provides as an example of a good mission statement the one that is used by OpenOffice.org (http://www.openoffice.org/): "To create, as a community, the leading international office suite that will run on all major platforms and provide access to all functionality and data through open-component based APIs and an XML-based file format." Fogel highlights following advantages of this mission statement: "In just a few words, they've hit all the high points, largely by drawing on the reader's prior knowledge. By saying "as a community", they signal that no one corporation will dominate development; "international" means that the software will allow people to work in multiple languages and locales; "all major platforms" means it will be portable to Unix, Macintosh, and Windows. The rest signals that open interfaces and easily understandable file formats are an important part of the goal. They don't come right out and say that they're trying to be a free alternative to Microsoft Office, but most people can probably read

- State That the Project is Free
    - Fogel (2007) strongly demands that “the front page must make it unambiguously clear that the project is open source.” He points out that many projects forget to do this, which may result in loosing many potential developers and users. Fogel recommends to “state up front, right below the mission statement, that the project is "free software" or "open source software", and give the exact license.”
- Features and Requirements List
    - A brief list of the features the software supports and the kind of computing environment required to run the software would help people who are interested in he software to understand what the software does and under which conditions it can be used. If features that are not completed yet are earmarked as "planned" or "in progress", (potential) users and developers will also see how the software will probably develop in future.
- Development Status
    - A development status page, listing the project's near-term goals and needs, informs people about how a project is doing in terms of development, maintenance, releases, and responsiveness towards bug reports, etc., and whether the software is available as alpha and/or beta version.9 Fogel points out that the development status page must not look ready.
- Downloads
    - You should offer interested people to download the software as source code in standard formats. According to Fogel (2007), binary (executable) packages are not necessary when a project is first getting started (except for software with so complicated build requirements or dependencies that merely getting it to run would be a lot of work for most people, which Fogel recommends to avoid anyway because it it creates a strong obstacle towards attracting developers).
    - The distribution mechanism should be as convenient, standard, and low-overhead as possible.
    - Software should comply with standard build and installation methods, as deviations from these standards would create confusion and difficulties for potential users and developers. As Fogel points out, this is something to be thought about very early, and not only when the code is close to being ready.
    - In this context, Fogel recommends that in general “boring work with a high payoff should always be done early, and significantly lowering the project's barrier to entry through good packaging brings a very high payoff.”
    - Using unique version numbers for different releases is important for people to compare any two releases and know which supersedes the other. (see also the chapter on release management below).

between the lines. Although this mission statement looks broad at first glance, in fact it is quite circumscribed: the words "office suite" mean something very concrete to those familiar with such software. Again, the reader's presumed prior knowledge (in this case probably from MS Office) is used to keep the mission statement concise.”

[9] “The term alpha usually means a first release, with which users can get real work done and which has all the intended functionality, but which also has known bugs. The main purpose of alpha software is to generate feedback, so the developers know what to work on. The next stage, beta, means the software has had all the serious bugs fixed, but has not yet been tested enough to certify for release. The purpose of beta software is to either become the official release, assuming no bugs are found, or provide detailed feedback to the developers so they can reach the official release quickly. The difference between alpha and beta is very much a matter of judgement.” (Karl Fogel (2007): Producing Open Source Software - How to Run a Successful Free Software Project. Available online at http://producingoss.com/en/index.html.)

- Version Control and Bug Tracker Access
    - Those users / developers who want to debug or add new features to your software need information about bug reports and bug fixes and how the code is changed over time. Real-time access to the latest sources is provided by a version control system, and an overview of bugs and fixes is provided by a bug tracking system. Bug trackers often also track enhancement requests, documentation changes, pending tasks, and more. Fogel considers the existence of these tools as so important that he highly recommends to clearly state on your website that you intend to set them up soon if you can't offer them right away.
- Communications Channels
    - Provision of the addresses of mailing lists, chat rooms, and IRC channels, and any other forums where others involved with the software can be reached enables visitors to get in touch with the persons involved with the project. Fogel recommends to make it clear that you and the other authors of the project are subscribed to these mailing lists, so people see there's a way to give feedback that will reach the developers. He points out that your presence on the lists does commit you to answer all questions or implement all feature requests.
    - In the early stages of a project user and developer forums can be integrated in order to have all discussions in one "room."
- Developer Guidelines
    - Those visitors who are interested in contributing to the project will look for developer guidelines. As Fogel points out, “developer guidelines are not so much technical as social: they explain how the developers interact with each other and with the users, and ultimately how things get done.” Basic elements of developer guidelines are:
        1. pointers to forums for interaction with other developers
        2. instructions on how to report bugs and submit patches
        3. some indication of how development is usually done—is the project a benevolent dictatorship10, a democracy, or something else
- Documentation
    - Something that is essential but lacking or provided in low quality in many Open Source Software projects is documentation. The problem is, according to Fogel, that documentation is never really finished.”
    - The most important documentation for initial users informs those about the basics: how to quickly set up the software, an overview of how it works, perhaps some guides to doing common tasks. A problem might occur because these things are known very well by the writers of the documentation, which makes it sometimes difficult for them to see things from the reader's point of view, and to laboriously spell out the steps that (to the writers) seem so obvious as to be unworthy of mention. (Fogel 2007)
    - Fogel recommends to use a simple, easy-to-edit format such as HTML, plain text, Texinfo, or some variant of XML, as this helps not only to remove any overhead that might impede the original writers from making incremental improvements, but also

[10] Fogel explains this term as follows: “No pejorative sense is intended by "dictatorship", by the way. It's perfectly okay to run a tyranny where one particular developer has veto power over all changes. Many successful projects work this way. The important thing is that the project come right out and say so. A tyranny pretending to be a democracy will turn people off; a tyranny that says it's a tyranny will do fine as long as the tyrant is competent and trusted.”

    those who join the project later and want to work on the documentation.
  - Basic initial documentation should meet the following minimal criteria (taken from Fogel 2007):
    1. “Tell the reader clearly how much technical expertise they're expected to have.
    2. Describe clearly and thoroughly how to set up the software, and somewhere near the beginning of the documentation, tell the user how to run some sort of diagnostic test or simple command to confirm that they've set things up correctly. Startup documentation is in some ways more important than actual usage documentation. The more effort someone has invested in installing and getting started with the software, the more persistent she'll be in figuring out advanced functionality that's not well-documented. When people abandon, they abandon early; therefore, it's the earliest stages, like installation, that need the most support.
    3. Give one tutorial-style example of how to do a common task. Obviously, many examples for many tasks would be even better, but if time is limited, pick one task and walk through it thoroughly. Once someone sees that the software can be used for one thing, they'll start to explore what else it can do on their own—and, if you're lucky, start filling in the documentation themselves. Which brings us to the next point...
    4. Label the areas where the documentation is known to be incomplete. By showing the readers that you are aware of its deficiencies, you align yourself with their point of view. Your empathy reassures them that they don't face a struggle to convince the project of what's important. These labels needn't represent promises to fill in the gaps by any particular date —it's equally legitimate to treat them as open requests for volunteer help.
    5. The last point is of wider importance, actually, and can be applied to the entire project, not just the documentation.
- Availability of documentation
  - Documentation should be available online (directly from the web site) and in the downloadable distribution of the software.
  - Online documentation should include a link to the entire documentation in one HTML page in order to allow people searching for a specific word or phrase across the entire documentation.
- Developer documentation
  - Developer documentation helps programmers to understand the code, so they can repair and extend it.
  - Fogel considers developer documentation to be very helpful, but he sees no reason to delay a release to do it. “As long as the original authors are available (and willing) to answer questions about the code, that's enough to start with. In fact, having to answer the same questions over and over is a common motivation for writing documentation. But even before it's written, determined contributors will still manage to find their way around the code” (Fogel 2007).
  - According to Fogel, wikis only work if the wiki is actively edited by a few people who agree on how the documentation is to be organized and what sort of "voice" it should have.
- Example Output and Screenshots
  - If the project involves a graphical user interface or if it produces graphical or

otherwise distinctive output, Fogel recommends to put some samples up on the project web site. For interfaces he advises to use screenshots; for output, screenshots or just files might be useful. One of the advantages of screenshots is that they convincingly demonstrate to the visitor of the website that the software really runs.
  - While not necessary for the starting period, in the future it might be useful to add also features like a news page, a project history page, a related links page, a site-search feature, a donations link, etc.
- Canned Hosting
  - You can leave hosting the project and the software to web services that offer free hosting and infrastructure for open source projects: a web area, version control, a bug tracker, a download area, chat forums, regular backups, etc. The advantage is that you get existing infrastructure for free (with huge server capacity and bandwidth), the disadvantage is that you loose fine-grained control over the user experience. The hosting service decides what software the site runs, and may control or at least influence the look and feel of the project's web pages (Fogel 2007).11
  - In the case of the public sector, the CIRCA IG OSS and Statistics and the OSOR (Open Source Observatory and Repository) - currently under development at the European Commission (Informatics Directorate) – help facilitating and making more visible links to all public sector projects in Europe.

In the field of Open Source Software for public sector institutions, some specific aspects of “marketing” and distributing the software must be considered:

- Enlarge the user’s base. Main target is always public administrations, but any use for any users should be admitted (this transparent and non-discriminatory distribution does not exclude private or commercial use, possibly outside the European territory)
- Respond to public sector policies and requirements (a growing number of government policies insist on the necessity to access the source code of government software)
- Provide the source code in order to authorise adaptations to local requirements, improvements that will benefit to the security, to the quality of the application and to its interoperability with other applications
- Encourage the integration of pre-existing OSS software modules (when licensed with compatible licences)
- Increase volunteers developers community, and therefore the potential resources to provide advises, to ensure support and product evolution: perennial character at long term, independently from support and financing;
- Develop a service market (an “ecosystem”) around the software: encourage service companies, having equal access to the code, to propose more competitive related services.

[11] “The largest and most well-known hosting site is SourceForge. Two other sites providing the same or similar services are savannah.gnu.org and BerliOS.de. A few organizations, such as the Apache Software Foundation and Tigris.org, give free hosting to open source projects that fit well with their missions and their community of existing projects” (Fogel 2007). Websites of the services mentioned by Fogel are http://sourceforge.net/index.php; http://savannah.gnu.org/; http://www.berlios.de/; http://www.apache.org/; and http://www.tigris.org/.

3.3 Decide how you want to distribute your Open Source Software (Licensing issues)

3.3.1 General licensing issues

If a copyright owner wants to distribute his/her work to others, he/she can sell the rights to it outright (known as “Assignment of Copyright”) or grant-limited rights to the work (known as a “Licence”). As with physical property, assigning the copyright involves handing over complete ownership of the work to a buyer or assignee. Another approach to exploit a copyright work is to licence it. A licence is a permission given by the copyright owner (known as the licensor) to another person (known as the licensee). The copyright owner agrees to allow the licensee to take actions that would otherwise be prohibited by law, such as copying, adapting and/or distributing the work. The licensee will agree to take these actions within the boundaries set by the licence — e.g. only creating and distributing a certain number of copies, or paying a royalty on each copy distributed. Although a licence can be created without stipulating a contract, most licences are granted in a legal form where both parties undertake certain obligations in respect of each other and the licensed copyright work; this is known as a “Licence Agreement”.[12]

Open Source licensing is the core element for any Open Source software.[13] Open Source licenses intend to promote wide distribution of software, and to encourage people who receive the software to contribute to its functionality by modifying the source code.[14] The main reason for a high number of licence models, when the same fundamental freedoms are present in all licences, is related to the modalities of re-distribution and to the control that the original author intends to keep of his or her work. In these fields, the diversity is nearly without limit. Luckily, only a few licences are really significant in terms of the number of projects published under them, or their potential: the choice should be made from a group of 5 to 6. The most important distinction is between three main families of licences: copyleft, permissive, and intermediary licenses.

[12] Licence Agreements are technical legal documents, which have many legal rules describing and confining their content and manner of expression. It is therefore worth noting that should there ever be a disagreement over whether there has been a breach of the agreement, in the final analysis the law will determine the outcome. A poorly drafted licence may then be found to mean something quite different from what both the copyright owner and the licensee intended, when construed by a court.

[13] According to the Open Source Definition of the Open Source Initiative (see http://www.opensource.org/docs/osd), an Open Source licence must: a) grant the licensee the right to distribute the program him/herself, including the right to charge money for it; b) grant access to the program's source code; c) grant the right to modify the program; d) grant the right to distribute modified versions of the program; e) allow the use of the program by all persons or groups in all fields of endeavour; f) apply to everyone who receives the program, without the need for any additional agreement; g) apply to the program it licences whether the program is obtained as part of a group of programs, or on its own; h) allow distribution with any other software; i) allow distribution in any form.

[14] A list of Open Source Licenses is provided at http://www.opensource.org/licenses. Although it may seem that granting these rights could lead to a large number of slightly variant versions of a piece of software, in practice successful Open Source software tends to absorb disparate modifications made by many contributors back into a single modified and improved version. It is a common misconception that one cannot charge money for distributing Open Source software. This is not the case. In practice, it is rarely done, mainly because each “re-distributor” could give the software away in direct competition with the original licensor. It is notable that the most widely-used Open Source licence — the GNU General Public Licence (GPL) — imposes a major further condition upon those who copy, adapt or distribute software under it: all software created by modifying the original software must also be licensed under the GPL (if it is licensed at all). It is worth noting that there is no compulsion to release changes that are made: either under the GPL or any other Open Source licence, the subsequent developer may keep secret internal versions of the modified software without necessarily licensing them to anyone else.

Copyleft licenses:

An Open Source licence is “copyleft” (also called “Share alike”) when it imposes on each subsequent “re-distributor” the obligation to distribute the result as a whole as Open Source, and in specific cases to reuse exactly the same licence. The most well known example is the GNU GPL (GNU General Public Licence V 2) supported by the Free Software Foundation (FSF). The EUPL licence adopted by the European Commission is also “copyleft” because it imposes to reuse specific open source licences, while leaving much more freedom to developers than the GPL (see hereafter the “compatibility” section). Other examples are the open software licence (OSL v 2.1 or v 3.0) the Eclipse licence V 1.0, the Creative Common licence v 1.0 or the CeCill licence V 2.0.[15]

Permissive licenses:

Differently from copyleft licences, permissive licences (also called “free for all”) impose few constraints in the case of re-distribution. In particular, there is no “copyleft” obligation, meaning that a proprietary software developer can include the distributed OSS components in his products. A typical permissive licence is the BSD licence; other examples are X-11 and other X-type licences[16] as Xfree86, the Apache software licence[17], the Cryptix General Licence[18], The W3C Software Notice and Licence[19], The Python Copyright Licence[20], the Zope Public Licence[21], the LDAP public licence[22] and the Phorum Licence[23] just to mention a few.

Intermediary licenses:

There is an intermediary type, also called ‘keep open”, where modifications to the software distributed under such licences have to be made available under the same licence as well, while larger works incorporating the software are not subject to the same conditions (meaning that they can also be distributed under permissive licences, or even as proprietary software). The LGPL (for GNU Lesser General Public Licence) used for Linux system libraries is one of these.[24] The Mozilla Public Licence (MPL) provides the same compromise between Open Source development and business.

Unfortunately it is not the case that all the source code that is available under an Open Source licence can be adapted and combined without restriction in order to produce new Open Source

[15] Under some of these licences, the original author (called “Initial developer) has the possibility to control the source code evolution. Sometimes also, the “obligation to reuse the same licence” is limited to this source code that “must be published / available” to the community and to the initial developer after each contributor's modification. Nevertheless, the binary executable version can be re-distributed separately under another licence (including proprietary ones), and this limits duplication by “non-contributors”. Because these two copyleft licences are not compatible with each other, OSL and GPL components cannot be integrated. The initial choice of a non-GPL copyleft licence, even if it seems interesting at first sight, is not generally recommended if the project owner wants to benefit from the contributions of the large GPL community.

[16] http://www.x.org/xdownload.htm

[17] http://www.apache.org/licence-1-1

[18] http://www.cryptix.org/docs/licence.html

[19] http://www.w3.org/consortium/legal/copyright-software-19980720

[20] http://www.python.org/doc/copyright.html

[21] http://www.zope.org/resources/zpl

[22] http://www.openldap.org/software/release/licence.html

[23] http://www.phorum.org/licence.txt

[24] The LGPL is identical to the GNU GPL in many ways, but distinguishes two different kinds of situations when one uses a library. A “work based on the library” means either the library itself or any derivative work under copyright law, while a “work that uses the library” means a program that contains no derivative of any portion of the Library but is designed to work with the library by being compiled or linked with it. This includes also proprietary software that could run inside the Open Source environment. Therefore the use of such licences is quite specific.

software. Two licences, which each meet the requirements of the Open Source Definition individually, may nevertheless contain terms that make them incompatible with each other. The GNU GPL provides an example of this: it mandates that code, which incorporates GPL-licensed code, must itself be licensed under the GPL as a whole. It also mandates that no additional restrictions on the rights it grants can be imposed on GPL-licensed code. These two conditions combine to mean that GPL-licensed code can only be merged easily with other GPL-licensed code, or with code whose licence imposes only conditions present in the GPL. The Free Software Foundation, who administers the GPL, makes available a list of licences that they consider to be compatible with it. Often the simplest way of resolving licence conflicts is to ask the code's author(s) if they would be willing to re-licence their code for inclusion. However this approach is only really practical where the number of authors is relatively small. Here also, the EUPL innovates by accepting that EUPL-licensed code could be merged with code from any "compatible licence" and publishes therefore a non-revocable list of compatible licences.

3.3.2 Dual licensing

It is worth noting that the full copyright holder always has the freedom to licence his work under several different licences, depending on the goal and the use (e.g. proprietary or not). This is called "dual licensing". This provides an option when reciprocal licenses, such as the GPL or EUPL1, have been chosen to distribute software that is owned by a public institution and if it is a policy goal to allow third party developers to appropriate derivatives of the original product owned by the public body. Dual licensing involves releasing the software to the general public, including all other users and developers, under a reciprocal licence for the benefits of such a licence. In addition, the software is available for licensing to developers who want to create proprietary derivatives, under separate, non-open-source (or "private"), licensing terms. These could be commercial terms involving royalty payments, or allowing the public body to buy derived works at a discount. They could also simply require that the developer notify the public body before making proprietary derived works. Indeed, dual licensing is a strategy adopted by firms, which release software they fully own, the most successful example being MySQL (which releases software under the GPL to the public, and under proprietary licences to other firms that want to make proprietary changes to its popular database engine).

3.3.3 Practical issues

The choice of a distribution licence should be done very early as it is a central piece to define the rights of contributing developers and as it is difficult to change the initial choice afterwards. In so far the licensor want to avoid any exclusive "appropriation" of its software by a clever third party (and we assume that this position is in general the one of the public administrations). Fogel (2007) recommends that "when choosing a license to apply to your project, if at all possible use an existing license instead of making up a new one. There are two reasons why existing licenses are better:

- "Default License":For Eurostat, the license to choose is the EUPL[25].
- Exceptions: If in special cases – mostly because of clauses on derivative work – in underlying software the EUPL is not feasible, another OSS licensing scheme accepted by OSI[26] can be chosen. In this case, the agreement of Director General of Eurostat is required.
- Familiarity. If you use one of the three or four most popular licenses, people won't feel they have to read the legalese in order to use your code, because they'll have already done so for that license a long time ago.

25 European Union Public License, version 1.1; see http://www.osor.eu/eupl
26 www.opensource.org

- Quality. Unless you have a team of lawyers at your disposal, you are unlikely to come up with a legally solid license.

(...)
Once you've chosen a license, you should state it on the project's front page. You don't need to include the actual text of the license there; just give the name of the license, and make it link to the full license text on another page. This tells the public what license you intend the software to be released under, but it's not sufficient for legal purposes. For that, the software itself must contain the license. The standard way to do this is to put the full license text in a file called COPYING (or LICENSE), and then put a short notice at the top of each source file, naming the copyright date, holder, and license, and saying where to find the full text of the license."[27]

A very important issue that should be considered when a license for distributing the software is chosen is security. In this regard, Open Source distribution as visible code is a two-edged sword. The good side is that it will multiply "reviser's eyes" and facilitate the discovery and rapid correction of bugs, including security failures. The bad side is that publication of un-secure software could facilitate illicit use or intrusion in your system prior to applying corrections.

Therefore the public authority should take its distribution decision after due consideration of the software functionalities: if misuse could facilitate identity theft, cyber terrorism or other forms of IT-enabled crime, Public Sector should consider the level of security as a main evaluation criteria and audit the work prior to distribution to avoid that the wrong people could gain access to confidential or/and sensitive information (including any kind of privacy protected and personal data).

The case of un-personal statistical software and the very complex calculations included is very different: open source code distribution will give the entire specialised user community the possibility of dynamically contributing to closing and concealing errors and security gaps. Disclosing the code will proactively enhance the software's quality and security level, representing all benefits for the reliability of the resulting information.

3.3.4 Tracking intellectual property

The issues of licence compatibility and of complex multiple ownership of intellectual property mean that it is desirable, if not essential, for programmers and their managers to keep detailed records. Version Control Systems provide some of this record keeping automatically, recording who made changes to the code and what they did. To complement this information, managers should keep records of the contractual and licensing status of contributors in order to establish who owns their work. They should also require and store explicit agreements from copyright owners that their contributions may be licensed and distributed under the licence selected for the project as a whole. Where code is brought in from existing Open Source software, the details of the relevant licence must be recorded (having first established that this licence is compatible with the project's overall licensing policy).

[27] Karl Fogel (2007): Producing Open Source Software - How to Run a Successful Free Software Project. Available online at http://producingoss.com/en/index.html. Fogel points out that there are many variations of this pattern.

4 Technical infrastructure

OSS communities typically use a number of collaboration technologies in order to coordinate work and validate the produced code and documentation (and the like). Such technologies should also be provided in statistical OSS projects, though the fact that many of the developer communities in this field remain very little may reduce the need of some of these technologies (such as IRC or sprints) as compared to other OSS projects. These technologies include:[28]

IRC (Internet Relay Chat), for group communication in chat rooms and also one-to-one communication for private talks. There is a chat room available in CIRCA, which is called "virtual meeting".

- Mailing lists, for discussions of people who are interested in the same topic; closed lists where the administrator has to add the recipients manually can be distinguished from public lists where people can subscribe on their own by sending an email to a specific address. An online mailing list archive with search functionality allows to group the conversations into the discussion threads.
- Wiki, a website where everyone can view, add and modify its contents.
- Blog, a web application that manages the frequent posting of news messages, which is used as the personal diary of an author categorizing his entries into different topics. There are also project-specific Blogs where various writers can post articles or a project hosts a so called Planet where all the messages of the contributors' Blogs are aggregated bearing the OSP's category. Most Blogs offer the possibility to add public comments so they act as an interactive medium between the authors and the readers. With protocols like RSS (Really Simple Syndication) the news can be monitored with RSS clients and they can also be referenced among the different Blog websites.
- Sprints, i.e. meetings of OSS developers for two or three days with the goal to intensify further development of the software in an XP (Extreme Programming) styled manner which includes pair programming among other things.
- Revision control systems in order to record all changes of the source code. They allow multiple developers to work on one software project simultaneously and ensure no file revisions get lost. Various systems are in use, of which the most commonly known are CVS (Concurrent Versions System) and Subversion (SVN).
- API (Application Programming Interface) refers to the overall structure and design of a software. An API provides a set of routines, protocols and tools for constructing software applications. In object oriented environments the API specifies how the behaviour and state of classes and objects is accessed. It defines what routines are available, how to call them, and what they do, but it does not explain how the subroutines are implemented and what algorithms are applied. The API tells thus developers how to access e.g. core functionality of a software framework in order to write their own application.

The European Statistical System (ESS) provides a number of additional requirements to be met by statistical OSS development projects in order to comply with European harmonisation goals. The most fundamental requirements in this respect are probably

- to use common standards for data processing (i.e. the same algorithms for the same tasks in different statistical institutions or applications)
- to use standardised statistical methodologies

[28] This part has been taken from the XDIS-OSS Guidelines on Developing Statistical Software as Open Source,

- to use a standardised architecture
- to use a common grammar in order to express validation rules
- to use standardised interfaces
- to use XML for data exchange between different applications (and SDMX for the exchange of aggregated statistical data)
- to use open standards in order to secure interoperability between different technical platforms
- to localise software in order to make it usable in as many languages of EU Member States as possible

5 Releasing

5.1 Recommendations

In contrast to proprietary software projects, open source software projects lack of centralized control over the development team, which has a strong impact on release managment. As Fogel (2007) points out, volunteer groups are not so monolithic but their members work on the project for all sorts of reasons. As a consequence, not all of them are interested in helping with a given release. Fogel (2007) describes how the community deals with the challenges deriving from the multitude of interests and the fact that software development is a continuous process (in which it can be hard to determine when a new release should be issued): “The model that makes this possible generalizes to more than just releases. It's the principle of parallelizing tasks that are not mutually interdependent—a principle that is by no means unique to open source development, of course, but one which open source projects implement in their own particular way. (...) They gravitate toward processes that have flat, constant levels of administrative overhead, rather than peaks and valleys. Volunteers are generally willing to work with small but consistent amounts of inconvenience; the predictability allows them to come and go without worrying about whether their schedule will clash with what's happening in the project. But if the project were subject to a master schedule in which some activities excluded other activities, the result would be a lot of developers sitting idle a lot of the time—which would be not only inefficient but boring, and therefore dangerous, in that a bored developer is likely to soon be an ex-developer.”

Open Source Software is distributed in packages and, usually, in different releases over time. In order to organise this process in an effective manner and to keep others' interest in using / advancing your software, you should comply with following recommendations:

- Release Numbering
  - Fogel (2007) describes the typical features of a release as follows:
    - old bugs have been fixed
    - new bugs have been added
    - new features may have been added
    - new configuration options may have been added
    - incompatible changes may have been introduced
  - Release numbering serves two purposes, it should unambiguously communicate the ordering of releases and it should indicate the degree and nature of the changes in the release. In order to make sure to achieve these goals, it should be kept to following recommendations:
    - Release numbers are groups of digits separated by dots, such as OpenOffice.org 2.4.0, whereby the dots are (usually) not decimal points but separators

        - Additional components of the release number are descriptive labels such as "Alpha" or "Beta", which means that this release precedes a future release that will have the same number without the qualifier
        - Other qualifiers in semi-regular use include "Stable", "Unstable", "Development", and "RC" (for "Release Candidate"). The most widely used ones are still "Alpha", "Beta", and "RC", and in order to ease understandability of the release number it is recommended to keep to digits and theses three additional labels.
        - Given this “signalling system”, release numbers inform the user exactly about the degree of changes and the status of the distributed software: an increment of the major number indicates that major changes happened; an increment of the minor number indicates minor changes; and an increment of the micro number indicates really trivial changes (Fogel 2007).
        - There are different conventions for versioning and release numbering, as, for instance, used by the APR project (http://apr.apache.org/versioning.html), but the differences between these conventions are quite marginal.
- Release Branches
    - Due to the fact that development is a continuous task it might appear difficult to determine when a formal new release should be issued.
    - It is not recommendable to take a snapshot of the tree at a moment in time, package it up, and hand it to the world as a new release because it may be hard to decide whether or not the entire development tree is clean and ready for release: newly-started features could be lying around in various states of completion, or someone might have checked in a major change to fix a bug, but the change could be controversial and under debate at the moment the snapshot is taken (Fogel 2008).
    - In order to overcome these problems one should always use a release branch.[29]
    - How exactly to create a release branch depends on the version control system that is used in a project.
    - The decision on which changes will be in the release and which will not, and shaping the branch content accordingly, is not an easy task. One problem is that a new release often creates a “last-minute feature rush (...): as soon as developers see that a release is about to happen, they scramble to finish their current changes, in order not to miss the boat. This, of course, is the exact opposite of what you want at release time (...). The more changes one tries to cram into a release at the last minute, the more the code is destabilized, and (usually) the more new bugs are created” (Fogel 2007).
    - According to Fogel (2007), most software engineers agree on a set of rough criteria for what changes should be allowed into a release line during its stabilization period:
        - fixes for severe bugs, especially for bugs without workarounds
        - documentation updates
        - fixes to error messages (except when they are considered part of the interface and

[29] Fogel (2008) defines a release branch as follows: “A copy of the project, under version control but isolated, so that changes made to the branch don't affect the rest of the project, and vice versa, except when changes are deliberately "merged" from one side to the other (see below). Branches are also known as "lines of development". Even when a project has no explicit branches, development is still considered to be happening on the "main branch", also known as the "main line" or "trunk".

Branches offer a way to isolate different lines of development from each other. For example, a branch can be used for experimental development that would be too destabilizing for the main trunk. Or conversely, a branch can be used as a place to stabilize a new release. During the release process, regular development would continue uninterrupted in the main branch of the repository; meanwhile, on the release branch, no changes are allowed except those approved by the release managers. This way, making a release needn't interfere with ongoing development work.”

    must remain stable)
    - certain kinds of low-risk or non-core changes to go in during stabilization, and may have formal guidelines for measuring risk

- Decision-making
  - A usual way to ease decision-making is to agree on one person to be the release owner (e.g. as a “benevolent dictator”); this does however not abandon discussions and arguments. The release owner should have the technical competence to understand all the changes, and the social standing and people skills to navigate the discussions leading up to the release without causing too many hurt feelings.
  - Alternatively, the project team can opt for a voting mechanism, i.e. vote on which changes to include in the release. The problem here is that the most important function of release stabilization is to exclude changes, therefore it's important to design the voting system in such a way that getting a change into the release involves positive action by multiple developers. Including a change should need more than just a simple majority (Fogel 2007).
- Eurostat release policy
  - The OSS paradigm states “Release early, release often!” This means, that the project officer should not wait for a very final bug-free version, but should publish beta versions as soon as they are available. The advantage is the chance to get early feedback from volunteer beta testers.
  - When publishing beta versions, please mark them clearly as such, and do not forget to mention known bugs and issues in the release notes. Do not publish beta versions, which are in such a bad state that a publication could burn the credibility of the project.

5.2 Source Code

The source code must be freely accessible (through an http/ftp link) without any user restrictions. This can be in achieved by a public CIRCA group, or a so-called “forge”; the best known forge is SourceForge[30], the recommended forge for the European Statistical System is OSOR[31].

When OSOR is chosen for the distribution, please make sure to activate the option, that all users have to accept the EUPL license before downloading the software.

If the application consists of several components, each component must be in the respective folder and which carries the name of the component

Internally each subfolder could be structured in a different way: by source type (code, images, text) or by function (core, plug-ins)

It is preferable that, at the top level of the source code, there is a script that automatically compiles the source code (when possible, it is convenient to use a multiplatform script language, like Ant). The source code should contain a copyright clause as follows:

```
© 2009 by the European Community, represented by Eurostat.
All rights reserved.
```

[30] http://www.sourceforge.net
[31] http://www.osor.eu

The year “2009” shall be replaced with the relevant year of creation. Authors (programmers) may be mentioned, companies having worked on the source code may only be mentioned in brackets after author’s names. All intellectual property rights of the code produced by contractors is with the European Community; there must not be any text in the source code diverging from this. Modules should not be given names referring to companies.

5.3 Documentation

Once the source code is released it is convenient to explain how to compile it in binary code and clarify whether additional software is needed.

Release the documentation of the source code: there are open source tools that automatically generate dependency diagrams starting from the source code (Doxygen, Javadoc, etc.)

Since the application can be modified the documentation must be released in an editable format (*txt, html, doc*[32], *odf/odt*).

The documentation can be in a separate folder or together with the source code. In the latter case, it is common practice to name the folder as “doc”.

5.4 Packaging

Open Source Software is distributed as source code, regardless of whether the software runs in source form (i.e., can be interpreted, like Perl, Python, PHP, etc.) or needs to be compiled first (like C, C++, Java, etc.). Compiled software will usually be installed from pre-built binary packages.

Despite of some projects deviating from this, there is a fairly strict standard for how source releases should look:

- Format
  - The source code should be shipped in the standard formats for transporting directory trees. For Unix and Unix-like operating systems, the convention is to use TAR format,[33] compressed by compress, gzip, bzip or bzip2. For MS Windows, the standard method for distributing directory trees is zip format, which happens to do compression as well, so there is no need to compress the archive after creating it (Fogel 2007).
- Name and Layout
  - the name of the package should consist of the software's name plus the release number, plus the format suffixes appropriate for the archive type.
  - the source code should be arranged in a layout ready for compilation (if compilation is needed) and installation
  - there should be a plain text README file explaining what the software does and what release this is in the top level of new directory tree, where also pointers to other resources, such as the project's web site, other files of interest should be provided, such as

[32] While Microsoft’s *doc* format is proprietary, it is sufficiently well supported by non-proprietary applications. The Open Office XML format (*docx*) however should at the moment still be avoided, as it is not sufficiently supported by OSS applications.

[33] “TAR stands for "Tape ARchive", because tar format represents a directory tree as a linear data stream, which makes it ideal for saving directory trees to tape. The same property also makes it the standard for distributing directory trees as a single file. Producing compressed tar files (or tarballs) is pretty easy. On some systems, the tar command can produce a compressed archive itself; on others, a separate compression program is used” (Fogel 2007).

        - INSTALL file and README file, both giving instructions on how to build and install the software for all the operating systems it supports
        - COPYING or LICENSE file, giving the software's terms of distribution
        - CHANGES file (sometimes called NEWS), explaining what's new in this release
- Specifications for handling statistical Open Source Software at Eurostat:
    - When distributing the software at the end there will be one folder containing source code, programmer's documentation, user guide, compilation and installation guide, etc.
    - If the application is intended to run on PCs, there should be a second folder containing the compiled binaries, user guide (unless there is an online-help), and installation instructions.
    - If applicable, further folders may be added.
    - Each folder shall contain a copy of the text of the EUPL plus a file *licence.txt* with the following content (replace XXXX by the name of the application and 2009 by the applicable year):

```
XXXX is © 2009 by the European Community, represented by
Eurostat. All rights reserved.
XXXX is published under the European Union Public License
(EUPL) version 1.1. If you do not accept this license,
you are not allowed to make any use of XXXX or any parts
thereof.
```

    - Where applicable, it is useful to add an MD5 file to check the integrity of the software once it is downloaded.
    - The folders may be compressed for easier download; it is common practice to compress them in *tar.gz* or in *zip* format.
    - It is common practice to name these files with the name of the application followed by the version number (e.g. 1.3, 2.0).
    - If the application is distributed together with third party components to ease the installation be the users, please check:
        - that the licensing conditions of the third party components do not prohibit this packaging;
        - that a *readme.txt* file must be included declaring all licenses the different components;
        - that the text of all licenses is included in the package

5.5 Installer

For applications running on a PC, it is possible to use an open source installer, which can include the EUPL license during the installation. The recommended installer is Inno Setup. In this case, the preferred distribution for is a single exe file. Make sure that the installation procedure is flexible enough to allow the user to specify an alternative destination drive and directory.

If the software does not necessarily require the installation to run properly, it is recommended that both the self-installing exe file and the zip file containing the binaries without the installation are offered for download.

6 Good practices for using the OSOR forge

The OSOR forge[34] is the recommended place to distribute applications. The OSOR forge is organized by project. A project may be a single application or a group of closely related applications.

In OSOR, everybody may read or download, but writing and uploading is reserved to registered and authorized persons. So, before creating a project in OSOR, you have first to register. Then you can create a project. Projects should be created by Eurostat officials rather than by contractors. After creation of the project, write authorization can be given to the contractor's personnel.
When registering a project in OSOR, do not indicate a website or a CDE page, if you want your project to be hosted by OSOR. Choose SVN for source code versioning. Do not forget to indicate the topic "Statistics".

The project officer decides the extent to which the OSOR forge is used. As a minimum, it has to be used for the distribution of the files. A typical release of an application could look like follows:

Filename	**Content**
`AppName-v2.3.tar.gz`	Source code; documentation in editable format; scripts for compilation, installation; etc.; *license.txt*, *EUPL.pdf*
`AppName-v2.3.zip`	Binaries and all other files needed to run the application; *license.txt*, *EUPL.pdf*
`Setup-AppName-v2.3.exe`	Single file for the installation on a Windows PC, a message to accept the EUPL is displayed at the start of the installation procedure

Release notes shall be included with each release after the first, using the OSOR release note feature.

Further tools on OSOR which may be used are the following:

- Forums: This is a discussion forum feature comparable to the CIRCA newsgroups, but more user friendly. It is advisable to use the feature, as it is easy to maintain.
- Tracker: This feature can be used for tracking bugs, support requests, patches and feature requests. It is recommended to make use of this feature, unless another incident tracking system has been agreed with the contractor.
- Lists: When a project is created, a mailing list is created automatically. Registered OSOR users may subscribe to this mailing list. This feature is intended to be used in collaborative projects between many partners (like ESSnets); otherwise it may be ignored.

[34] http://forge.osor.eu/

- Tasks: This is a To-do-list feature, it should only be used in a collaborative project with several partners and a project co-ordinator committed to maintain the lists.
- Docs: This is the place for general documentation on the application, including background documents. Please note that user and program documentation should stay with the code.
- Surveys: In a large and diverse community, surveys may be a way to find a good agreement for future roadmaps etc. For Eurostat, this feature should be used with care: in many cases IPM could be the better solution.
- News: A service for announcements. Please announce new releases and other important news here.
- SCM: This is the file storage for the source files. When the contractor does not have his own version management system for the source code, this should be used. In any case, the source code must be uploaded here at the end of a contract, so that a new contractor will be able to take over responsibility.
- Files: This is the download section to be used for the distribution of the files. The usage of this part of the OSOR forge is mandatory for all projects having already released a version. A typical release of an application could look like follows:

Filename	**Content**
`AppName-v2.3.tar.gz`	Source code; documentation in editable format; scripts for compilation, installation, etc.; *license.txt, EUPL.pdf*
`AppName-v2.3.zip`	Binaries and all other files needed to run the application; *license.txt, EUPL.pdf*
`Setup-AppName-v2.3.exe`	Single file for the installation on a Windows PC, a message to accept the EUPL is displayed at the start of the installation procedure

Release notes shall be included with each release after the first, using the release note function at upload time.

- Wiki: The Eurostat project officer has to decide, if the project needs its own Wiki, or if the “OSS and Statistics” Wiki in the OSOR communities section[35] or the planned Wiki on Eurostat’s homepage are the better choice.

7 Making use of the OSOR community “OSS and Statistics”

The project officers are invited to join the OSOR community “OSS and Statistics” (http://www.osor.eu/communities/oss-and-statistics). They should keep the wiki updated on their project, and should inform the facilitators of the community on new releases, so that the community pages are updated, and publicity can be made via the “news” feature.

[35] http://www.osor.eu/communities/oss-and-statistics/wiki

8 Helpful software

- Apache Ant: http://ant.apache.org/ (scripting language for compilation)
- Doxygen: http://www.stack.nl/~dimitri/doxygen/ (documentation tool)
- Javadoc: http://java.sun.com/j2se/javadoc/ (documentation tool)
- Kompozer: http://kompozer.net/ (HTML editor)
- Inno Setup: http://www.innosetup.com/ (installer tool)
- HashCheck: http://www.ktechcomputing.com/hashcheck/ (hash code tool)
- 7Zip: http://www.7-zip.org/ (packer)